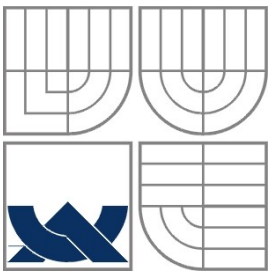
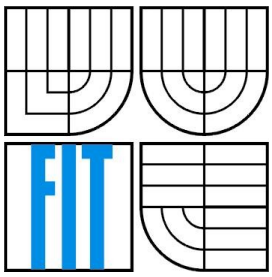


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ALGORITMY PRO VYHLEDÁNÍ NEJDELŠÍHO SHODNÉHO PREFIXU

LONGEST PREFIX MATCH ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAROSLAV SUCHODOL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JIŘÍ TOBOLA

BRNO 2010

Abstrakt

Práce se zabývá směrováním v IP sítích, konkrétněji otázkou zjištění nejdelšího shodného prefixu. Problematiku vyhledání nejdelšího shodného prefixu řeší mnoho sofistikovaných algoritmů. Hlavním úkolem této práce je zaměření na následující algoritmy - Controlled Prefix Expansion, Lulea Compressed Tries, Binární vyhledávání na intervalech a Binární vyhledávání na prefixech. Algoritmy jsou principiálně popsány a následně softwarově implementovány v jazyce Python. Výstup práce spočívá v analýze/porovnání jednotlivých algoritmů z hlediska paměťové náročnosti a počtu přístupů do paměti v nejhorším případě.

Abstract

This work deal with routing in IP networks, particularly the issue of finding the longest matched prefix. Problems of finding the longest matched prefix solves many sophisticated algorithms. The main task of this work focuses on the following algorithms - Controlled Prefix Expansion, Lulea Compressed Tries, Binary search on intervals and Binary search on prefix length. Algorithms are described and followed by software implementation in Python. Output work is the analysis/comparison of different algorithms in terms of memory consumption and number of memory accesses at worst.

Klíčová slova

IP, směrování, nejdelší shodný prefix, Controlled Prefix Expansion, Lulea Compressed Tries, Binární vyhledávání na intervalech, Binární vyhledávání na prefixech

Keywords

IP, routing, Longest Prefix Match, Controlled Prefix Expansion, Lulea Compressed Tries, Binary search on intervals, Binary search on prefix length

Citace

Jaroslav Suchodol: Algoritmy pro vyhledání nejdelšího shodného prefixu, bakalářská práce, Brno, FIT VUT v Brně, 2010

Algoritmy pro vyhledání nejdelšího shodného prefixu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jiřího Toboly. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jaroslav Suchodol
19.5.2010

Poděkování

Děkuji vedoucímu práce panu Ing. Jiřímu Tobolovi za odbornou pomoc a vlídnost při konzultacích.

Obsah

1 Úvod

S rozmachem počítačových sítí a především internetu narůstá komunikace mezi lidmi, kterých je připojeno k internetu čím dál více. Tato komunikace v dnešní době nezahrnuje jenom nenáročné posílání textových zpráv, ale také novější techniky dorozumívání přes tuto „síť sítí“, jako jsou například hlasové, či video přenosy, které jsou náročnější na přenos dat. Také u prohlížení webových stránek vznikl požadavek na větší přenosy dat, jelikož tyto stránky často nabízejí online pouštění hudby, či videí. Jsou tedy zapotřebí náležité techniky a zařízení, aby se tak objemné operace mohly uskutečnit.

Cílem této práce je jedna z nezbytných technik, pro přenos dat skrze počítačové sítě, která se nazývá vyhledání nejdelšího shodného prefixu. Tato technika určuje kam budou data dále směrována, při průchodu určitým síťovým zařízením, s ohledem na nejefektivnější možnou cestu. Detailnější popis této techniky a směrování v IP sítích nalezneme v kapitole 2.

Na kapitolu 2 dále navazuje kapitola 3, ve které jsou popsány vybrané algoritmy zabývající se problematikou vyhledání nejdelšího shodného prefixu. V kapitole 4 zaměřené na implementaci práce vysvětluji výhody programovacího jazyka Python, popis činnosti výzkumné skupiny ANT a nakonec samotné detaily implementace. Předposlední, 5-tá kapitola představuje každý algoritmus zvlášť, z pohledu paměťové náročnosti a počtu přístupů do paměti v nejhorším případě. Poslední 6-tá kapitola porovnává paměťové nároky implementovaných algoritmů.

2 Podstata směrování v IP sítích

2.1 IP Protokol

IP je anglická zkratka pro Internet Protocol, představený v [1] a definovaný v RFC 791. Již podle názvu můžeme usoudit, že má tento protokol něco společného s Internetem. A opravdu tomu tak je, poněvadž tento standard tvoří základní protokol dnešního Internetu. IP je datový protokol navržený pro použití v propojených počítačových sítích založených na paketově orientovaném přenosu dat. Slouží k přenášení datových bloků známých jako datagramy z místa jejich vzniku do místa určení, přičemž jsou tyto místa identifikována pomocí adresy o pevné délce (IP adresa). V každém datagramu je uvedena IP adresa odesílatele a příjemce.

Předtím než zahájíme komunikaci není potřeba navazovat spojení s druhou (cílovou) stranou či jinak „připravovat cestu“ datům. Tím pádem je IP takzvanou nespolehlivou službou. Tato vlastnost má za následek, že datagramy vůbec nemusí dorazit do cíle, či dojít vícekrát, neručí se ani za správné pořadí doručených paketů. Pokud by se pakety často ztrácely či přicházely v jiném pořadí, mělo by to za následek snížení výkonu sítě, pozorovatelné uživatelem. Ovšem příležitostná ztráta/chyba nemívá pozorovatelný efekt. Doručování datagramů je tedy prováděno jako best effort - „nejlepší úsilí“. Respektive, data která chceme poslat přes síť, využívající IP, jsou rozdělena na datagramy a ty poslány do sítě. Tyto datagramy putují sítí zcela nezávisle jeden na druhém. Všechny zařízení na trase datagramu, usilují podle svých možností o poslání datagramu blíže k jeho cíli.

IP taktéž poskytuje fragmentaci a opětovné sestavení datagramů. Což je nezbytné v situacích, kdy má být velký paket přenesen přes síťové zařízení, které s takto velikým paketem neumí pracovat. Velikost datagramu je proměnná, může být od 20 bajtů (20 bajtů hlavička a 0 bajtů dat) až po 65 535 bajtů (64 KB).

2.1.1 Verze IP

Internet Protokol se během své působnosti vyvíjel a přinášel nové verze označované jako IPvX [1], kde X je číslo verze protokolu. Verze 0 až 3 jsou rezervované nebo nepoužité, verze 5 je experimentální. V současné době se tedy IP vyskytuje ve dvou verzích a to ve verzi 4 a 6, tedy IPv4 a IPv6. IPv6 je nástupcem IPv4. Dále se budu zabývat pouze verzí 4, tedy pod pojmem IP adresa, budu mít vždy na mysli IP adresu verze 4.

IP adresa se používá v počítačových sítích k jednoznačnému určení právě jednoho síťového zařízeni, je 4 bajty (oktety) neboli 32 bitů dlouhá. Díky tomu může IPv4 adresovat až 2^{32} (přibližně 4 miliardy adres) zařízení, některé adresy jsou ale vyhrazené pro speciální účely (například multicast), tudíž se počet reálně použitelných adres zmenší. I když se zdá, že takové množství adres může

postačovat, opak je pravdou. S rozvojem Informačních Technologií a Internetu je čím dál více zařízení připojeno do sítí s IP adresováním, proto ubývá IPv4 adres.

Řešením problému s nedostatkem IP adres by měla být nová verze protokolu, již zmíněná IPv6, která se ovšem světem rozšiřuje velice pomalu. Protože je ne vždy podporována zařízeními/softwarem a je mnohem méně pohodlná na práci než její předchůdce IPv4. IPv6 je 128 bitů dlouhá a nabízí tak větší adresní prostor než IPv4, konkrétně 2^{128} možných adres, což je z dnešního pohledu nevyčerpatelné množství.

IPv4 adresa má čtyři částí (bajty), tyto části jsou vytvořeny pouze z číslic od 0 do 9 v možném rozsahu jednoho bajtu, tedy 0 až 255. Kdežto adresa IPv6 je uspořádána z osmy skupin (2 bajty). Skupinu tvoří číslice 0 až 9 a navíc i písmena A, B, C, D, E, F. Tak dosáhneme většího množství adres. Následuje ukázka těchto adres.

Ukázka adresy IPv4: 158.234.16.109
Ukázka adresy IPv6: 2001:0470:1F0A:3AF1:4322:FEC4:90DC:ACDC

2.1.2 Dělení IP adresy

IP adresa (32 bitů) je logicky rozdělena na dvě části [2]: „síť“ a „host“. Toto rozdělení je nezbytné, jestliže chceme určit kam patří/směřuje daná IP adresa. Síťová část identifikuje síť do které patří daná IP adresa a druhá část „host“ nám říká, jaké konkrétní zařízení v této síti chceme adresovat.

Které bity v adrese přísluší síťové či hostové části nejsou pevně dané. Rozdělení se odvíjí podle velikosti sítě, respektive kolik zařízení chceme adresovat. Při vytváření sítě si naplánujeme jak chceme aby byla naše síť velká. Tedy jaké číslo a kolik bitů přidělíme síťové části. Síťová část má síťové číslo a síťovou masku indikující, které bity odpovídají síťové části.

Následuje obrázek pro demonstraci sítě 158.234.0.0, používající síťovou maskou 255.255.0.0, která nám umožní adresovat (256 * 256) - 2 zařízení v této síti. V tomto případě je prvních (nejlevějších) 16 bitů všech IP adres dané sítě shodných s 1001110 11101010. Tedy IP adresy této sítě budou začínat čísly 158.234.

síťová maska

11111111	11111111	00000000	00000000

síťové číslo

10011110	11101010	00000000	00000000

Obrázek 2.1: Síťové číslo a maska.

2.2 Směrování (routování)

Pojem směrování představený v [3] je označení pro hledání cest v počítačových sítích. Úkolem směrování je dopravit datový paket určenému adresátovi, co nejefektivnější cestou. Protože cesta paketu k jeho cíli může byt dlouhá, přes mnoho zařízení, musí tyto zařízení přeposílat paket dokud se nedostane do místa určení. Směrování se tak většinou nezabývá celou cestou paketu, ale řeší vždy jen jeden krok – komu data předat jako dalšímu.

Směrovací proces můžeme částečně přirovnat k cestě autem. Přijedeme ke křižovatce, podíváme se na tabuli se značením směru a podle ní se rozhodneme, kam dál pokračovat. Řekněme, že chceme jet do Brna, na tabuli si najdeme Brno a směr kam máme pokračovat (například vpravo). Zabočíme tedy doprava a neřešíme, jak budeme pokračovat na další křižovatce. Teprve až k ní dojedeme a prohlédneme si zdejší tabuli, rozhodneme kam dál pokračovat v jízdě (například doleva). A tak budeme postupovat i nadále, dokud nedorazíme do cíle. Podobně každé směrující zařízení (tabule z textu výše) obsahuje sadu pravidel (ukazatele směru), která říkají, kam se mají předávat pakety směřující k určeným cílům. Podle nich se data předají některému ze sousedních zařízení, kde se bude rozhodovací proces opakovat podle zdejších pravidel.

Směrování dat provádí většinou, ale ne vždy, zařízení známé jako směrovač. I ostatní zařízení na síti mohou směrovat, dokonce koncové počítače, byť v jejich případě bývá směrování triviální.

2.2.1 Směrovací tabulka

Základní datovou strukturou pro směrování je směrovací tabulka (routing table) [3], která má podobu elektronické tabulky nebo databáze, jež je uložena ve směrovacím zařízení. Představuje taktéž onu sadu ukazatelů, podle kterých se rozhoduje, co udělat s kterým paketem. Má tedy uložené směry k jednotlivým cílům sítě. Směrovací tabulka je složena ze záznamů obsahujících:

- *cílovou adresu*, které se dotyčný záznam týká. Může se jednat o adresu individuálního počítače, častěji však je cíl definován *prefixem*, tedy začátkem adresy. Prefix mívá podobu 129.15.0.0/16, která představuje rozsah IP adres (129.15.0.1 až 129.15.255.254) spadající do tohoto prefixu a zároveň určuje, že záznam s tímto prefixem je platný pro IP adresu z daného rozsahu. Hodnota před lomítkem je adresa cíle, hodnota za lomítkem pak určuje počet významných bitů adresy. Jak již bylo naznačeno uvedenému prefixu vyhovuje každá adresa, která má v počátečních 16 bitech (čili prvních dvou bajtech) hodnotu 129.15.
- *akci* určující, co provést s datagramy, jejichž adresa vyhovuje prefixu. Akce mohou být dvou typů: doručit přímo adresátovi (pokud je dotyčný stroj s adresátem přímo spojen) nebo předat některému ze sousedících uzlů (jestliže je adresát vzdálen).

Směrovací rozhodnutí se poté uskutečňuje pro každý přicházející datagram samostatně.

2.3 Princip vyhledání nejdelšího shodného prefixu

Vezmeme cílovou adresu datagramu a srovnáme ji se směrovací tabulkou. Z tabulky se vyberou všechny vyhovující záznamy (prefix vyhovuje cílové adrese). Z vybraných záznamů se použije ten s nejdelším prefixem. Toto pravidlo vyjadřuje přirozený princip, že konkrétnější záznamy (jejichž prefix je delší, tedy přesnější) mají přednost před obecnějšími.

Ku příkladu mějme paket jehož cílová IP adresa je 101.11.12.13 a snažíme se najít nejdelší shodný prefix v routovací tabulce, která obsahuje následující dva záznamy:

	„prefix sítě“	„následující uzel“
1)	101.11.0.0/16	101.11.1.5
2)	101.11.12.0/24	101.11.12.5

Zde vidíme, že jsou oba záznamy platné pro cílovou IP adresu, protože cílová IP adresa koresponduje s „prefixem sítě“ daných záznamů. Největší shodu má však záznam s „prefixem sítě“ 101.11.12.0/24, jelikož má nejdelší síťovou masku (/24), která je větší než jiné shodne prefixy (/16). Tudíž výsledek našeho hledání je ten, že paket pošleme na „následující uzel“ toho záznamu, který má největší shodu cílové IP adresy s „prefixem sítě“. Tedy na „následující uzel“, který má IP adresu 101.11.12.5.

3 Algoritmy pro vyhledání nejdelšího shodného prefixu

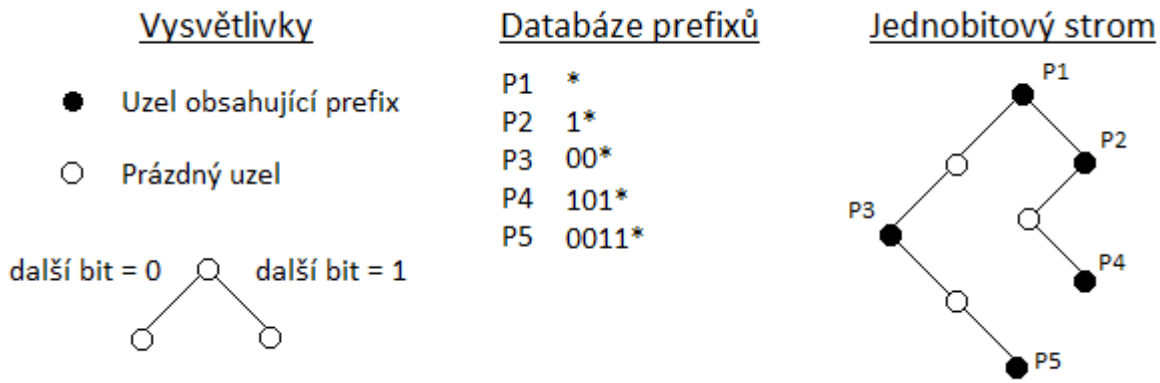
Algoritmus vyhledání nejdelšího shodného prefixu (Longest Prefix Match – LPM) má vstup množinu prefixů z routovací tabulky a cílovou IP adresu, pro kterou hledá cestu. Výstupem je pak nejdelší prefix, který vyhovuje cílové IP adrese.

Existuje mnoho algoritmů řešící tuto problematiku, já se budu ovšem zabývat pouze algoritmy, které jsem implementoval v rámci této práce a to Controlled Prefix Expansion, Lulea Compresed Tries, Binární vyhledávání na intervalech a Binární vyhledávání na prefixech. Předtím, než se budu věnovat těmto algoritmům, vysvětlím základní algoritmus Trie, z kterého vychází dva dále popsané algoritmy Controlled Prefix Expansion a Lulea Compressed Tries.

3.1 Trie

Trie algoritmus [4] má podobu binárního stromu (stromová datová struktura) složeného z uzlů , kde jsou uloženy prefixy z routovací tabulky. Nad tímto stromem se později vyhledává LPM s konkrétní IP adresou. Strom se začíná vytvářet od kořene (root), který reprezentuje prefix nulové délky (odpovídá všem adresám). Z uzlu vedou dva ukazatele na další uzly stromu (jestli je kam navazovat): "0" a "1". Každý uzel představuje právě jeden prefix (binární cesta od kořene do daného uzlu). Následuje ukázka pro tento algoritmus.

Mějme cílovou adresu 001100 (v binární podobě, protože budeme hledat v binárním stromu), pro kterou chceme najít LPM na již sestaveném stromu z prefixů na obrázku 3.1. Začneme hledat po jednom bitu od vrcholu stromu, podle bitů cílové adresy budeme postupovat stromem a ukládat prefixy, které najdeme cestou stromem v uzlech. Stromem postupujeme do té doby než, vyčerpáme všechny bity adresy nebo narazíme na konec stromu. Poslední nalezený prefix je nejdelší shodný prefix pro vzorovou adresu, tudíž výsledek algoritmu. Tedy P5 z obrázku pro adresu 001100.



Obrázek 3.1: Ukázka binárního stromu.

Výhodou [5] algoritmu je jednoduchá datová struktura, rychlá aktualizace vyhledávaných prefixů a nízké paměťové nároky. Nevýhodou je pak rychlost, která v nejhorším případě pro adresu IPv4 značí 32 náhodných přístupů do paměti.

Existují mnohá rozšíření tohoto algoritmu, která se zaměřují na urychlení výpočtu (zpracování více bitů současně), nebo efektivnější uložení datové struktury.

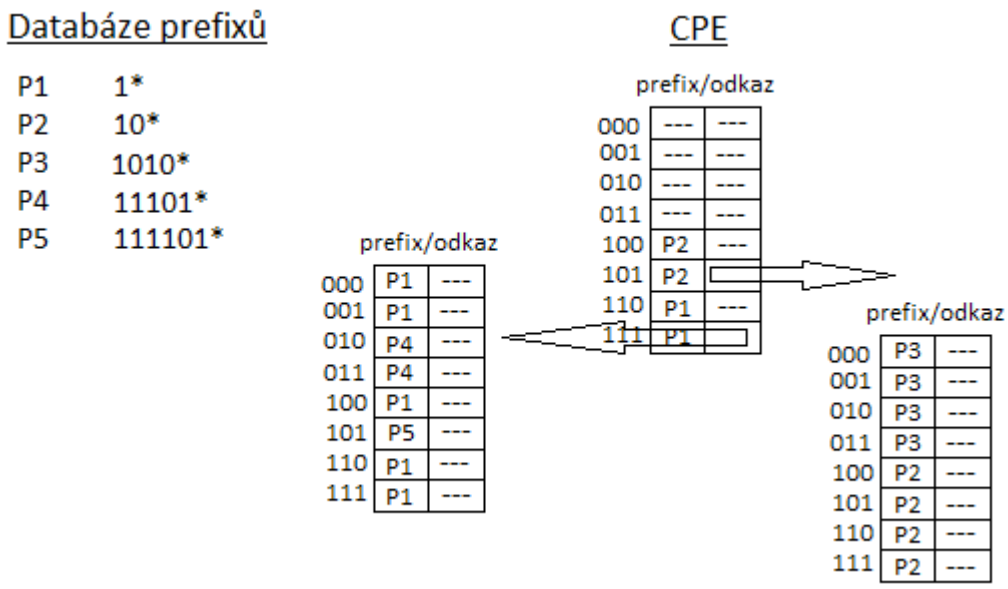
3.2 Controlled Prefix Expansion (CPE)

CPE [5] a [6] je algoritmus založený na algoritmu Trie. Algoritmus Trie zpracovává pouze jeden bit v čase, kdežto CPE jich může zpracovávat hned několik najednou (takzvaná střída), tudíž je rychlejší. Ovšem při použití velké střídy (přibližně větší než 10) vzniká mrhání pamětí. Musíme tedy vhodně zvolit použitou střídu. Problém nastává s prefixy, které nemají stejnou délku jako násobek střídy. Proto jsou všechny prefix rozšířeny na délku, která odpovídá násobku střídy. Ku příkladu při střídě 4 a prefixu 10*, který má délku 2, rozšíříme tento prefix na délku 4: 1000, 1001, 1010, 1011. Pokud již existuje skutečný prefix shodný s nějakým z expandovaných prefixů, použijeme skutečný prefix namísto expandovaného.

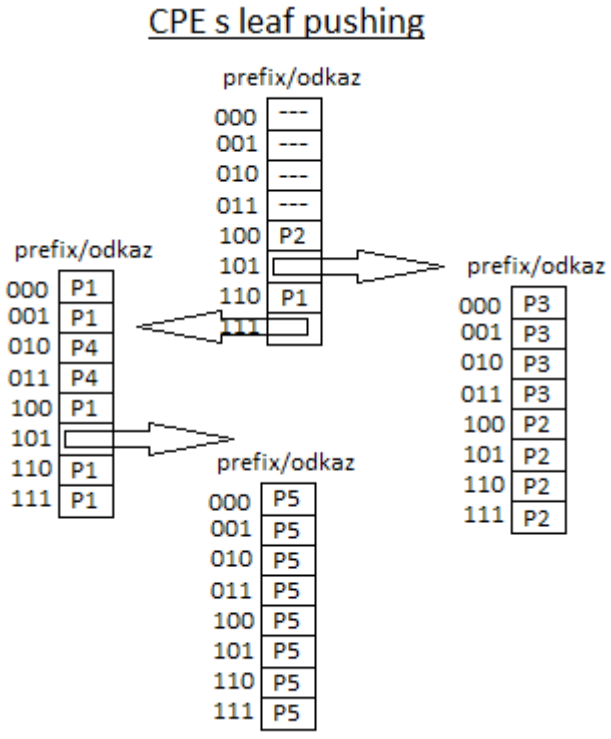
Každý uzel stromu obsahuje dvě pole, první pro uložení prefixů a druhé pro odkazy na další uzly, zpřístupněné pomocí 2^n (n je použitá střída) binárních klíčů. Jestliže v poli (polích) na dané pozici binárního klíče, není nic uloženo, nastává mrhání paměti. Proto vzniklo rozšíření algoritmu s anglickým názvem "leaf pushing", které se zaměřuje na úsporu potřebné paměti. Úspora paměti je 50%. Respektive místo dvou polí existuje jenom jedno společné pole, v kterém může být uložena na jedné pozici právě jedna hodnota prefixu nebo odkazu. Z této úpravy vyplývá, že prefix délky stejné jako násobek střídy, bude uložen v následujícím uzlu s plnou expanzí. Toto můžeme pozorovat u prefixu P5 na obrázku 3.3, kde se nachází ukázka CPE s „leaf pushing“, podle databáze prefixů z obrázku 3.2.

Následuje obrázek 3.2, na kterém jsou ukázány tři uzly stromu. Strom je sestaven z databáze prefixů na témže obrázku a střídou 3. Ve stromu se poté vyhledává jako u metody Trie, pouze s tím rozdílem, že v jednom kroku projdeme více bitů (rychlejší vyhledávání).

Existuje i alternativní [6] CPE algoritmus, který umožňuje mít různé počty zpracovávaných bitů v každém uzlu. Nicméně v hlavním uzlu musí zůstat pevná délka zpracovávaných bitů.



Obrázek 3.2: CPE strom.



Obrázek 3.3: CPE strom s rozšířením „leaf pushing".

3.3 Lulea Compressed Tries

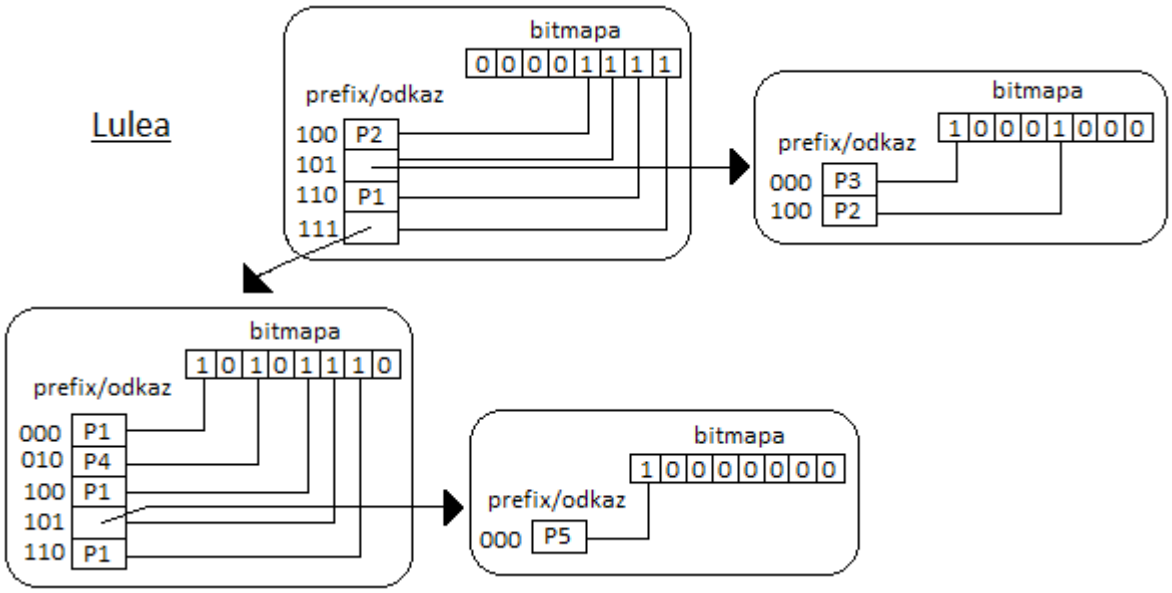
Lulea představená v [4] a [5] je další z mnoha rozšíření základního stromového algoritmu Trie. Navazuje na metodu CPE, ve směru redukce potřebné paměti. Může zpracovávat stejně jako metoda CPE více bitů najednou. Hlavní myšlenkou této metody je redukovat počet uložení prefixu, který je v uzlu uložen za sebou. Tím docílíme menší potřeby paměti v uzlech stromu.

Respektive, u algoritmu CPE, je při expandování prefixu a jeho následné uložení v uzlu velké plýtvání paměti. Nejvíce to pocítíme když bude prefix násobkem střídy, protože se prefix uloží v následujícím uzlu s plnou expanzí, což znamená, že budeme zbytečně ukládat jeden stejný prefix na 2^n pozic v uzlu. Proto u algoritmu Lulea vznikla nová část, která má podobu bitmapy, kde si poznačujeme, odkud začíná uložení prefixu a dále na jakých pozicích v uzlu je tento prefix platný. Přičemž první pozici odkud je prefix uložen označíme v bitmapě jako „1“ a ostatní pozice platné pro tento prefix jako „0“. Tím docílíme toho, že je prefix uložen v poli prefixů pouze jednou (v různých případech může být i vícekrát), ale přes bitmapu můžeme dohledat celou jeho platnost.

Vyhledávání ve stromu Lulea odlišujeme od toho v algoritmech Trie a CPE. I když je počáteční princip shodný s CPE (začneme od kořene stromu; směr udávají bity adresy; vyhledání skončí až v cestě nebudou další odkazy), je na konci cesty důležitá změna prováděné akce. Jestliže pozice v poli prefixů/odkazů, kde jsme skončili obsahuje prefix navrátíme jej jako výsledek. Pakliže konečná pozice neobsahuje prefix, ale v bitmapě existuje tato pozice, obsahující „0“, najdeme prefix tím, že půjdeme tak dlouho v bitmapě směrem k nižší pozici (na obrázku 3.4 je to směr vlevo), dokud nenajdeme „1“. Binární klíč pozice bitmapy, na které jsme našli „1“, koresponduje s pozicí (klíčem) v poli s prefixy/odkazy, zde se nachází vyhledaný prefix.

Jak již bylo naznačeno, je hlavní výhodou algoritmu dovednost smrštit uložení prefixu tak, že šetří pamětí. Nevýhodou pak náročnější přidání nového prefixu.

Příklad smrštění předvedu na prefixech uložených v uzlu napravo od hlavního (root) uzlu stromu nacházející se na obrázku 3.2. U CPE metody s „leaf pushing“ by v tomto uzlu byly uloženy prefixy/odkazy v následujícím pořadí: P3, P3, P3, P3, P2, P2, P2, P2. Díky smrštění stejných prefixů, které se nacházejí za sebou, dostaneme zredukovaný obsah uzlu, tedy: P3, P2. Zúžili jsme tak počet uložených prefixů v daném uzlu z 8 na 2. Aby bylo možné později správně vyhledávat LPM v uzlech stromu, musíme vytvořit bitmapu, která odpovídá smrštění pole s prefixy uzlu. Výsledná bitmapa bude mít podobu 10001000, na základě poznatků uvedených výše. Grafická reprezentace je na obrázku 3.4.



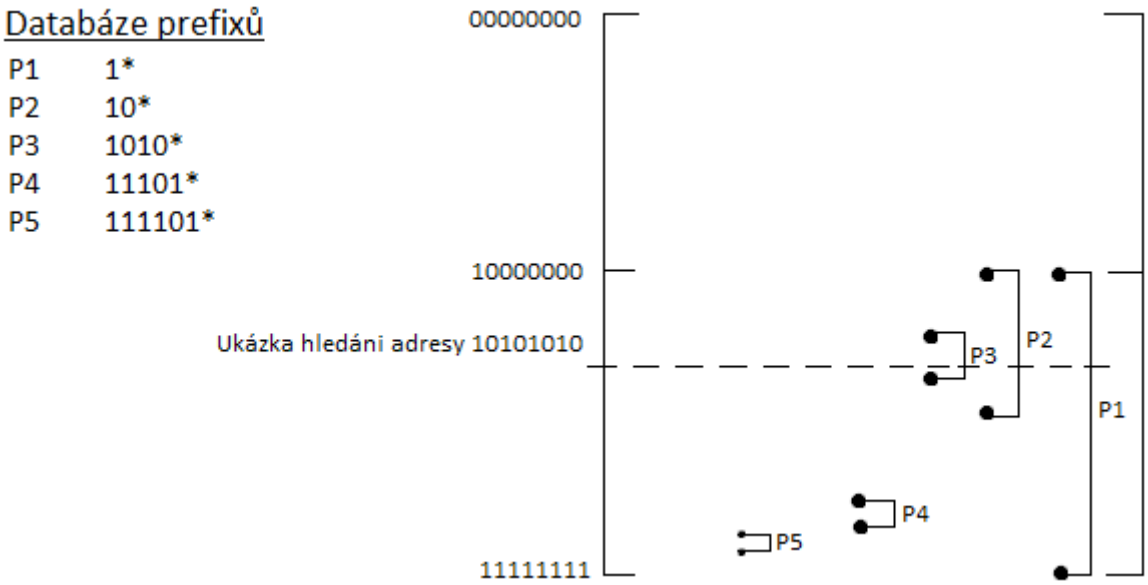
Obrázek 3.4: Podoba struktury Lulea, založena na databázi prefixů z obrázku 3.2.

3.4 Binární vyhledávání na intervalech

Tato metoda [5] a [7] je založena na poznatku, že je každý prefix interval, v kterém se nacházejí platné hodnoty (adresy) pro daný prefix. S tímto poznatkem vezmeme databázi prefixů, podle niž vytvoříme sadu intervalů. Z této sady dále utvoříme seřazenou tabulku, která nám umožní snadnější vyhledávání.

Algoritmus využívá k vytvoření intervalu expanzi prefixu, nulami a jedničkami. Tudíž začátkem intervalu bude zpracovávaný prefix rozšířený o samé nuly, a konec intervalu je tentýž prefix rozšířený o samé jedničky. Pro příklad vezměme prefix 1* a délku adresy 8 bitů, vzniklý interval bude začínat na hodnotě 10000000 a končit na 11111111. Takto rozšířené prefixy (intervaly) jsou poté vzestupně seřazeny do tabulky, nad kterou se později provádí vyhledávání nejdelšího shodného prefixu.

Na obrázku níže je vyobrazena grafická reprezentace intervalů, vytvořených z modelových prefixů na témže obrázku, které jsou shodné s těmi z obrázku 3.2. Můžeme si všimnout, že se grafický rozsah některých prefixů překrývá. To vypovídá o tom, že překrývající se prefixy mají společnou určitou část intervalu, v které se nacházejí platné adresy pro daný prefix. V případě hledání nejdelšího shodného prefixu pro adresu, která se nachází ve společném úseku pro více prefixů, je výsledný prefix ten, který má nejmenší grafický rozsah ze všech shodných prefixů. Ku příkladu P3 pro adresu 10101010, jak ukazuje obrázek 3.5.



Obrázek 3.5: Grafické zpodobnění prefixových intervalů.

Jak již bylo řečeno, data z obrázku můžeme převést na tabulku, v které se snadněji vyhledává. Prefixy jsou v tabulce seřazeny podle rozsahů. Vyhledání nejdelšího shodného prefixu provedeme tak, že postupně kontrolujeme jestli vyhledávaná adresa spadá do intervalu dvou sousedních hodnot v tabulce. Výsledek se pak nachází na řádku s první hodnotou, ze dvou sousedních hodnot kam spadá adresa. Tedy řádek s nižší pozicí.

Zkusme najít LPM pro adresu 11001100, za pomoci tabulky 3.6 vytvořené z prefixů na obrázku 3.5. Podle výše uvedených poznatků dospějeme k tomu, že adresa spadá mezi pozice 4 a 5. Výsledek je prefix na nižší pozici, tedy P1.

Výhodou [5] metody je velmi dobrý poměr potřebné paměti k použitým prefixům. Nevýhodou potom vkládání nového prefixu, protože musíme zachovat tabulku seřazenou, což při vkládání velkého množství prefixů může probíhat zdlouhavě.

Pozice	Hodnota	Prefix
1	10000000	P2
2	10100000	P3
3	10101111	P2
4	10111111	P1
5	11101000	P4
6	11101111	P1
7	11110100	P5
8	11110111	P1

Obrázek 3.6: Seřazená tabulka z expandovaných prefixů.

3.5 Binární vyhledávání na prefixech

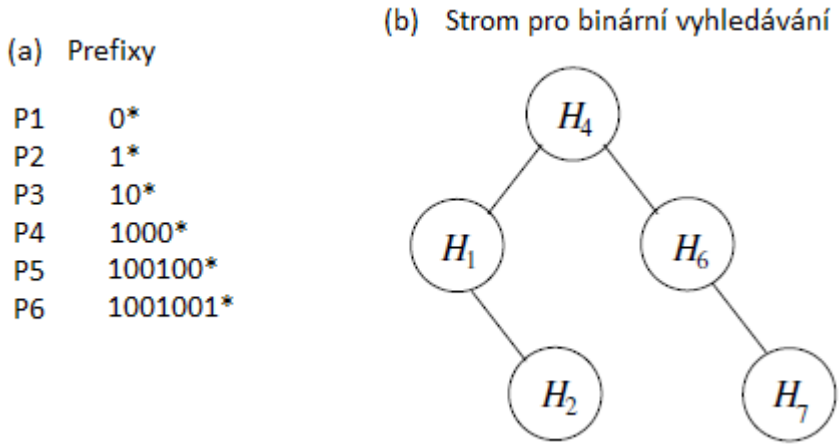
Algoritmus uvedený v [8] a [9] využívá metody Trie, respektive pracuje s datovou strukturou binárního stromu, která obsahuje uzly, ve kterých jsou uloženy potřebné informace pro chod algoritmu. Na rozdíl od metody Trie používá rychlejšího způsobu vyhledávání, takzvané hašování. Přesněji vezmeme data z hašovacích tabulek a vytvoříme podle nich stromovou strukturu, ve které budeme vyhledávat. Vyhledává se postupně od kořene stromu na prefixech stejné délky.

Abychom mohli sestavit strom musíme vybrat z množiny prefixů všechny délky daných prefixů, z těchto délek vytvoříme seřazený seznam. Střed seznamu zůstane (připojí se ke stromu) jako hlavní (nadřazený) uzel. Dále seznam rozpůlíme a se zbylou levou a pravou částí samostatně pokračujeme ve vytváření stromu (opakujeme nynější postup). Každý uzel poté obsahuje prefixy určité délky. Díky takovémuto vytváření stromu, můžeme později při vyhledávání, postupovat přímo k nižší či vyšší polovině stromu, při každém pohybu stromem. Což je oproti základnímu vyhledávaní nad jednobitovým stromem velká úspora kroků, pro najití toho co hledáme. Úsporu si můžeme představit jako najití čísla 74 v poli od 1 do 100. V binárním stromu (Trie algoritmus) se musíme postupně posouvat, po jedné pozici (od 1 do 74), než najdeme co hledáme. Kdežto u procházení stromu, který navazuje na hašovací tabulky můžeme skákat na určitá místa (podle algoritmu vytvoření stromu, který vytvoří specifické uzly), a tak najít výsledek rychleji.

Pro správný chod algoritmu jsou v každém uzlu uchovány následující informace:

- Délka (n významných bitů) uložených prefixů v tomto uzlu.
- Samotné prefixy s délkou n,
- k nim značky v případě, jestli existuje delší prefix, který má počáteční bity shodné s konkrétním prefixem v tomto uzlu.
- Značky (oříznutý delší prefix na délku n) které reprezentují delší prefixy v dalších uzlech
- a pro každou značku delšího prefixu její LPM.

Následuje příklad vytvoření stromu uvedený v [8]. Mějme množinu prefixů P1 až P6, jak ukazuje obrázek 3.7(a). Těchto 6 prefixů má 5 různých délek: 1, 2, 4, 6, 7. Podle postupu uvedeného výše vytvoříme strom, který je stejný s obrázkem 3.7(b). H_n (H – hašovací tabulka) reprezentuje právě jeden uzel, který obsahuje prefixy délky n, a další potřebná data uložená v uzlu zmíněná výše.



Obrázek 3.7: Strom sestavený z hašovacích tabulek.

Vyhledání nejdelšího shodného prefixu pro cílovou adresu bude probíhat následovně:

1) Začneme od kořene stromu (později v navazujícím uzlu), kde se podíváme jestli je prvních n bitů adresy shodných s nějakým prefixem, který je uložen v aktuálním uzlu. Jestliže je nějaký prefix shodný poznačíme si jej jako LPM pro danou adresu. Dále se podíváme jestli je u nalezeného prefixu umístěna značka existence „podobného“ delšího prefixu. Když je zde značka umístěna (pokračujeme v hledání), posuneme se na následující uzel vpravo (směrem k prefixům větší délky). Pakliže poznačený prefix neměl značku, skončíme hledání, jehož výsledek je právě poznačený prefix.
2) Pokud nebyl žádný prefix v uzlu shodný s adresou, zůstanou nám ještě dvě možnosti. První z nich je, pokusit se podobným způsobem jako u prefixů, o najití shody s adresou. Nyní však budeme hledat nad značkami „delších prefixů“. Jestliže bude nějaká značka shodná, označíme její LPM jako dosavadní výsledek. Poté pokračujeme v hledání na následujícím uzlu vpravo.
3) Poslední možnost, pokud se nenašla shoda bitů adresy u prefixů ani u značek „delších prefixů“, je posunout hledání na následující levý uzel, směrem ke kratším prefixům. A navrátit se k bodu 1.

S tímto principem postupujeme uzly, dokud nenajdeme výsledek nebo dojdeme na konec stromu.

Prospěch metody spočívá v nízkém počtu přístupů do paměti. Naopak nedostatek lze pozorovat ve vyšších paměťových nárocích.

4 Implementace

Veškerá práce byla vytvářena do existujícího projektu výzkumné skupiny ANT. Zhotovené části byly posílány do subversion[1] (SVN) skupiny ANT. Na přiloženém DVD se nachází celý repositář ze subversion skupiny ANT, tedy i implementované algoritmy. Zdrojový kód je komentovaný v anglickém jazyce, kvůli budoucímu využití.

4.1 Python

Implementace byla vytvářena softwarově v programovacím jazyce Python, konkrétně ve verzi 2.6.4. Python [10] je silný dynamický programovací jazyk, který se používá v celé řadě aplikačních oblastí. Je též částečně srovnatelný s Tcl, Perl, Ruby, Scheme nebo Javou. Mezi jeho klíčové znaky patří tyto vlastnosti:

- velmi jasná a čitelná syntaxe
- silné schopnosti introspekce
- intuitivní orientace objektu
- přírodní vyjádření procedurálního kódu
- plná modularita, podpora hierarchických balíčků
- zpracování chyb založeno na výjimkách
- velmi vysoká úroveň dynamických datových typů
- rozsáhlé standardní knihovny a moduly třetích stran pro prakticky každou úlohu
- snadné rozšíření a moduly napsané v C, C++
- doložitelný v aplikacích jako skriptovací rozhraní

4.2 Popis výzkumné skupiny ANT

Zdrojové kódy vzniklé při realizaci této práce budou využity výzkumnou skupinou Accelerated Network Technologies, působící na Fakultě Informačních Technologií VUT v Brně.

Skupina se zabývá [11] akcelerací časově kritických operací, které se používají v zařízení síťové infrastruktury a pro monitorování a bezpečnost sítí. Rychlost zpracovávání síťového provozu je důležitá ve většině síťových zařízení, neboť jakákoliv ztráta paketu může ovlivnit kvalitu služeb, má vliv na přesnost monitorování nebo může zamezit detekci útočníka. Současná řešení jsou buď drahá nebo mají omezenou rychlost zpracování. Výzkum skupiny je proto zaměřen na hledání nových algoritmů a architektur, které umožní konstrukci levnějších a rychlejších síťových zařízení a systému. Využíváme standardní i vícejádrové procesory, technologii FPGA a zaměřujeme se na operace

1 Systém pro správu a verzování zdrojových kódů

analýzy hlaviček paketů, klasifikace paketů, udržování a správa síťových toků, hledání nejdelšího společného prefixu a hledání řetězců nebo regulárních výrazů.

4.3 Jednotlivé algoritmy

Při realizaci jsem vycházel z principů a vlastností uvedených u jednotlivých algoritmů v kapitole 3. Při vytváření zdrojového kódu byla použita dekompozice problémů na podproblémy. Jenž značí funkce a třídy ve zdrojových souborech. Funkce a třídy jednotlivých algoritmů můžeme snadno použít, stačí jenom importovat soubor s algoritmem. Všechny implementované algoritmy převádí vstupní prefixy a IP adresu na binární podobu, kvůli nutnosti algoritmů pracovat s bity.

Součástí práce bylo taktéž implementovat IPv6 parser, který převádí IPv6 adresu do datové struktury, se kterou můžeme dále pracovat ve zdrojových kódech.

4.3.1 CPE

Zdrojový kód algoritmu obsahuje hlavní třídu CPE a dvě pomocné třídy Node, Tree. Všechny tři třídy mají jeden nepovinný argument, střídu, jestliže tento argument nenastavíme použije se výchozí hodnota, která je má velikost 3. Hlavní třída obsahuje několik funkcí pro práci nad metodou. Následuje popis jednotlivých funkcí.

Funkce report_memory vrací údaje o aktuálních paměťových nárocích. Konkrétně počet prefixů načtených do stromu, počet uzlů stromu, počet odkazů na následující uzly, počet odkazů na prefixy a největší hloubku stromu. Funkce load_prefixset vytvoří z množiny prefixů strom, do kterého uloží ony prefixy, k tomu je zapotřebí pomocná třída Node a Tree. Především pak funkce add_prefix ze třídy Tree, která uloží jeden prefix z množiny prefixů do stromu. Hlavní třída dále obsahuje funkce lookup a display. Funkce lookup provádí vyhledání LPM podle IP adresy, která je jediným argumentem funkce. Poslední funkce display zobrazí strukturu stromu.

Třída Node obsahuje seznamy pro uložení binárních klíčů, prefixů/odkazů a pomocný seznam (ke zjištění, zda je na určitém místě seznamu prefixů/odkazů, uložen prefix nebo odkaz). Třída Tree obsahuje funkci add_prefix se dvěma povinnými argumenty, první je prefix v binární podobě a druhý pak ukazatel na tento prefix.

Metoda je implementovaná včetně rozšíření „leaf pushing“.

4.3.2 Lulea

Při vytváření byl použit jako základ kód algoritmu CPE. Zdrojový kód algoritmu Lulea zahrnuje třídy LULEA, Node, Tree. Třída LULEA používá nepovinný argument, který značí střídu. Když tento argument nebudu nastaven, použije automaticky střídu o hodnotě 3. Do tříd Node a Tree také

vstupuje argument značící hodnotu střídu, avšak zde je povinný. Třída LULEA má stejné názvy funkcí jako u CPE – report_memory, load_prefixset, lookup, display. Tyto funkce vykonávají principiálně stejné činnosti ovšem nad Lulea strukturou.

Třída Node, která reprezentuje uzel, obsahuje slovníky pro uložení prefixů/odkazů, bitmapy a pomocného slovníku (který značí, jestli je určitá pozice v uzlu prefix či odkaz). Třída Tree, určená pro práci nad stromem, obsahuje funkci add_prefix se dvěma povinnými argumenty, první je prefix v binární podobě a druhý pak ukazatel na tento prefix. Tuto funkci odlišujeme od stejnojmenné funkce u CPE metody, především kvůli práci s bitmapou.

4.3.3 Binární vyhledávání na intervalech

Zdrojové kódy této metody obsahují pouze jednu třídu, pojmenovanou „BSI“. Ve které se opět nacházejí funkce společných názvů, jako ty z algoritmů již zmíněných. Funkce report_memory vrací počet správně načtených prefixů a velikost tabulky (počet řádků). Protože je ve zdrojovém kódu použita jenom jedna třída, odehrává se vložení prefixu do tabulky, a věci sním spojené (opětovné seřazení tabulky), ve funkci load_prefixset. Funkce lookup provádí vyhledání LPM, přes cílovou IP adresu, v tabulce. Poslední funkce display zobrazí seřazenou tabulku s již expandovanými prefixy.

Testování u této metody ukázalo znepokojivě pomalý chod algoritmu s velkým počtem vstupních prefixů. Po důkladné analýze zdrojového kódu byla zjištěna příčina na místě, kde se přidává nový prefix do tabulky. Konkrétně neefektivní procházení tabulkou, jenž bylo v dané chvíli na implementováno jako postupné, po jednom záznamu. Po nahrazení za „skokové“ procházení (přesun do určité části tabulky), bylo zvýšení rychlosti algoritmu zcela zřejmé.

4.3.4 Binární vyhledávání na prefixech

Poslední algoritmus má ve zdrojových kódech obsaženy dvě třídy, „BSP“ a Node. Primární třída „BSP“ používá opět stejnojmenné funkce jako u předchozích algoritmů: report_memory, load_prefixset, lookup, display.

Funkce report_memory vrací podobné údaje, jako totožná funkce u CPE. Jak bylo zmíněno v sekci 3.5, pracuje tato metoda se stromovou strukturou. Mohlo by se tedy očekávat, po vzoru CPE a Lulea, zahrnutí třídy Tree do kódu. Ovšem při implementaci se na tuto skutečnost pozapomnělo a kód, který měl být ve třídě Tree nalezneme ve funkci load_prefixset. Tato skutečnost nijak neovlivňuje správný chod algoritmu. Další funkce lookup slouží k vyhledání nejdelšího shodného prefixu. Zobrazení struktury algoritmu, přes funkci display, bylo vytvořeno podle vlastního uvážení, a proto se doporučuje budoucí úprava funkce.

Dosavadní algoritmy založené na metodě Trie používají třídu Node a tato metoda nebude výjimkou. Ovšem zde třída Node obsahuje největší množství dat (viz sekce 3.5), z doposud zmíněných tříd Node.

5 Simulace algoritmů

V této kapitole se zaměřím na simulaci jednotlivých algoritmů z pohledu paměťových nároků a počtu přístupů do paměti v nejhorším případě. Hlavní funkcí algoritmů je teoretická představa o náročnosti/rychlosti daného algoritmu, neboť praktická představa je po softwarové stránce špatně uchopitelná.

5.1 Paměťové nároky

Protože se tato práce zabývá pouze softwarovou implementací je neúčinné měřit spotřebu paměti u tříd a jiných implementačních pomůcek (zdrojové kódy jednotlivých algoritmů nemusejí efektivně pracovat s pamětí). Proto bude dále pojem paměťové nároky reprezentovat velikost celé struktury daného algoritmu, kterou zjistíme určením velikosti jednoho uzlu/záznamu v bajtech a následně za pomoci dat z funkce report_memory spočteme celkové paměťové nároky.

Simulace byly vytvořeny nad skutečnými routovacími záznamy z protokolu BGP[2] [12] pro IPv4. Poněvadž originální tabulka obsahovala příliš velké množství záznamů (322648), byly vytvořeny podmnožiny routovacích záznamů, přibližně o 1000, 10 000 a 50 000 záznamech. Tyto podmnožiny zachovávají rozložení prefixů. Byly vybrány pouze N-té řádky originální tabulky, kde N reprezentuje vhodně zvolenou konstantu (6, 32, 320).

5.1.1 CPE s „leaf pushing“

Uzel algoritmu obsahuje pole prefixů/odkazů adresované binárními klíči. Velikost pole závisí na použité střídě. Například při střídě 4 může pole uchovat 2^4 položek. Konkrétní paměťové nároky na jeden uzel se střídou 4 budou následující:

- 4 B pro jeden prefix/odkaz

Tedy plné pole zabere 4 B * 16 míst v poli = 64 B. Celkové paměťové nároky uvedu dvakrát, podle velikosti ukazatele:

1. Pevný ukazatel na prefix/odkaz, délka 32 bitů (4 B).
2. Nejmenší možný ukazatel na prefix a odkaz. Ku příkladu, pro 1009 prefixů postačuje 10 bitový ukazatel. A pro 6544 uzlů postačuje 13 bitový ukazatel.

Výpočet velikosti uzlů: ((počet odkazů na prefix + počet odkazů na další uzel) * 4 B) / 1000 = velikost všech uzlů (celého stromu) [kB]. Následuje příklad výpočtu pro střídu 2 a 1009 prefixů. Funkce report_memory vrátí mimo jiné pro tento vstup (různost podle použitých prefixů)

2 Border Gateway Protocol (BGP) [13] je dynamický směrovací protokol používaný pro směrování mezi autonomními systémy (AS). Je základem propojení sítí různých ISP v peeringových uzlech.

'prefix_pointers': 3550, 'child_pointers': 6543. Podle vzorce výše zjistíme velikost všech uzlů: ((3550 + 6543) * 4) / 1000 = 40,37 kB.

Výpočet velikosti uzlů s nejmenšími ukazateli: (počet odkazů na prefix * velikost nej. ukazatele na prefixy v bitech + počet odkazů na další uzel * velikost nej. ukazatele na uzly v bitech) / 8 / 1000 = nejmenší velikost všech uzlů (celého stromu) [kB]. Následuje příklad výpočtu pro střídu 2 a 1009 prefixů. Funkce report_memory vrátí mimo jiné pro tento vstup (různost podle použitých prefixů) 'prefix_pointers': 3550, 'child_pointers': 6543. Podle vzorce na začátku tohoto odstavce zjistíme nejmenší velikost všech uzlů: (3550 * 10 + 6543 * 13) / 8 / 1000 = 15,07 kB. Z výpočtu můžeme usoudit, že při zvýšení počtu 'prefix_pointers' a 'child_pointers', klesá rozdíl mezi celkovou pamětí (kvůli velikost nej. ukazatele, blížící se 32 bitům) pro ukazatel 32 bitů a nejmenší ukazatel.

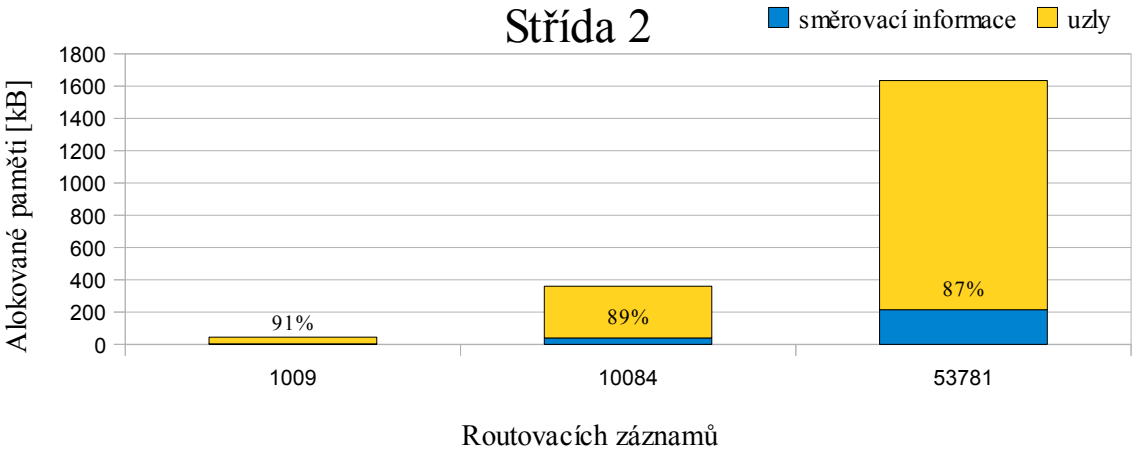
Protože se uzly skládají pouze z ukazatelů, můžeme vhodně zvolenou velikostí ukazatele ušetřit značné množství paměti.

Celková paměť reprezentuje součet velikosti uzlů se směrovacími informacemi. Podle výše uvedených postupů vznikla následující tabulka, která ukazuje paměťové nároky, nad různým množstvím prefixů, pro střídy 2, 4, 8. Střída 16 nemohla být kvůli svým obrovským paměťovým nárokům zařazena do testů.

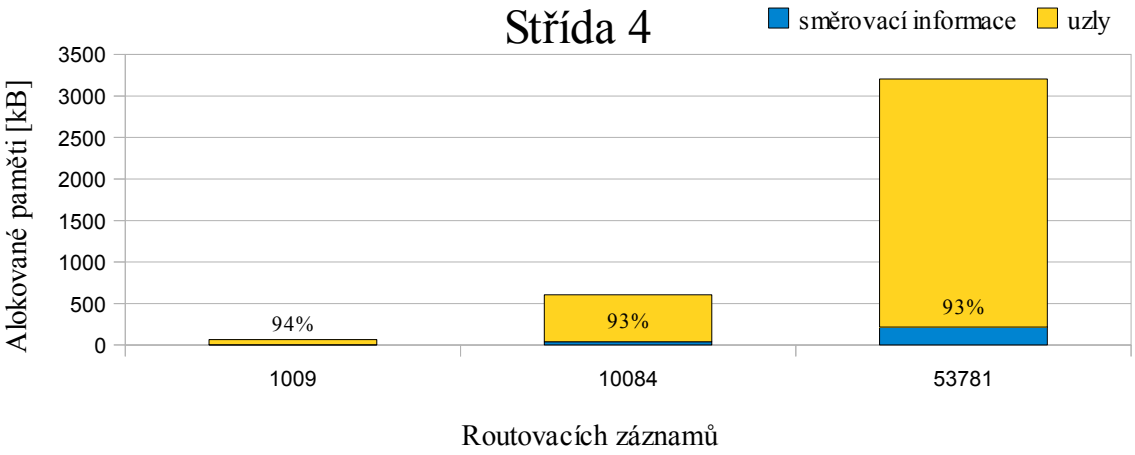
CPE s "leaf pushing"		Uzly		Celková paměť	
Směrovací informace	Střída	Počet uzlů	Velikost uzlů [kB]	Ukazatel 32 bitů	Nejmenší ukazatel
1009 adres / 4,04 kB	2	6544	40,37	44,41	19,11
	4	3381	59,98	64,02	23,63
	8	1705	609,6	613,64	194,75
10084 adres/ 40,34 kB	2	44358	320,4	360,74	191,61
	4	23203	564,76	605,11	290,33
	8	12666	6196,21	6236,55	2751,18
53781 adres / 215,12 kB	2	155082	1419,79	1634,91	963,79
	4	82759	2988,06	3203,18	1719,49
	8	43374	33394,93	33610,05	16912,58

Tabulka 5.1: Paměťové nároky CPE s rozšířením „leaf pushing“.

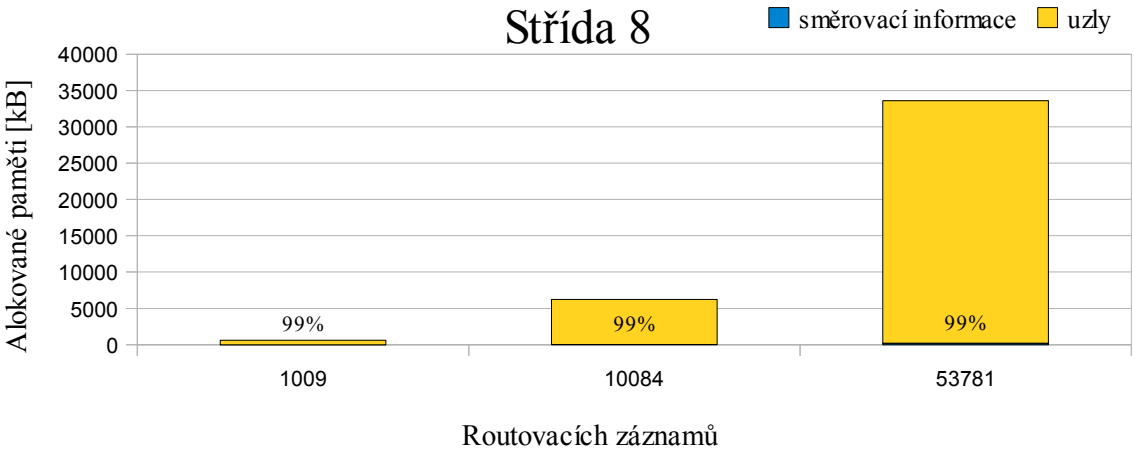
Z tabulky byly vytvořeny obrázky pro jednotlivé střídy, nad všemi směrovacími informacemi. Obrázky 5.2 – 5.4 zobrazují velikost alokované paměti pro celou strukturu s ukazateli 32 bitů. Tyto obrázky slouží také k demonstraci podílu „užitečných“ (směrovací informace) a „neužitečných“ (uzly) informací.



Obrázek 5.2: Velikost alokované paměti pro střídu 2 s rozdělením na směr. informace a uzly.



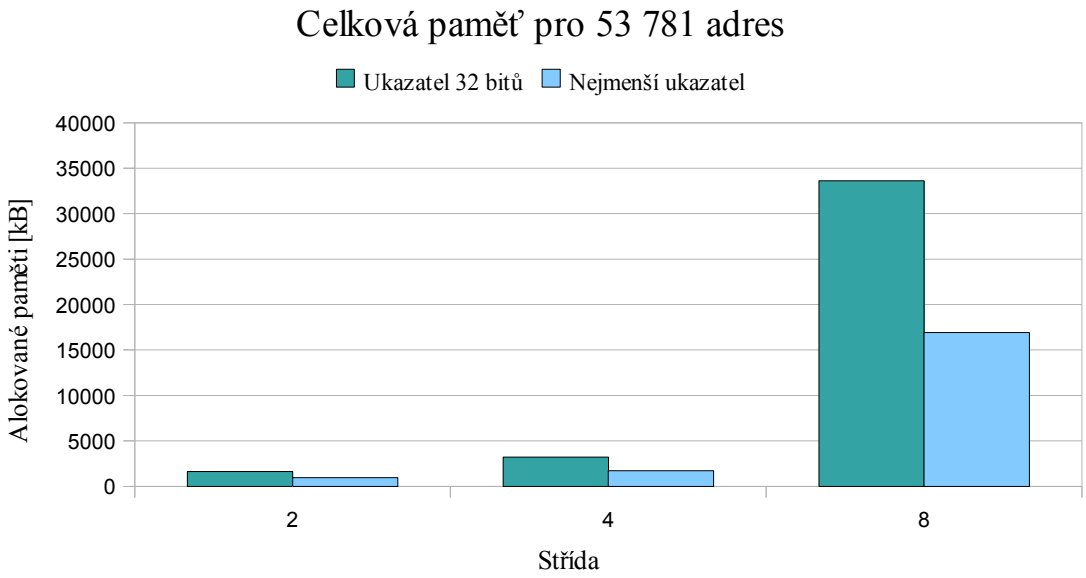
Obrázek 5.3: Velikost alokované paměti pro střídu 4 s rozdělením na směr. informace a uzly.



Obrázek 5.4: Velikost alokované paměti pro střídu 8 s rozdělením na směr. informace a uzly.

Z obrázků 5.2 – 5.4 vyplývá, že při zvyšující se střídě nastává větší mrhání paměti než u stříd nižších, protože většinu alokované paměti zabírají uzly (pro přehlednost jsou v obrázcích uvedena i konkrétní procenta). Dále můžeme z obrázků vyčíst nejlepší poměr nutných (směr. info.) / nadbytečných (uzly) dat u střídy 2.

Následuje obrázek 5.5 porovnávající celkovou paměť u stříd s pevným a minimálním ukazatelem. U stříd 2 a 4 můžeme pozorovat adekvátní paměťové nároky. Nesmíme však zapomenout, že se velikost uzlu odvíjí od mocniny dvou, to je důvod proč má střída 8 nepřiměřené paměťové nároky.



Obrázek 5.5: Velikost celkové alok. paměti pro pevné a minimální ukazatele, nad 53 781 prefixy.

5.1.2 Lulea

Uzel algoritmu obsahuje bitmapu a pole prefixů/odkazů. Velikost pole a bitmapy záleží na použité střídě. Konkrétní paměťové nároky na jeden uzel se střídou 4 budou následující:

- 4 B pro jeden prefix/odkaz
- 2 na střídu bitů, respektive 2 B pro střídu 4

Plné pole zabere 4 B * 16 míst v poli + 2 B = 66 B. Mohlo by se tedy zdát, že Lulea zabírá více paměti (kvůli bitmapě) než CPE. Ovšem u Luley je skutečná zaplněnost pole podstatně menší než u CPE. Následují výpočty paměťových nároků, které jsou podobné jako u CPE. Opět uvedu celkové paměťové nároky dvakrát, podle principu zmíněného u CPE.

Výpočet velikosti uzlů: ((počet odkazů na prefix + počet odkazů na další uzel) * 4 B + X B za jednu bitmapu * počet uzlů) / 1000 = velikost všech uzlů (celého stromu) [kB]. Následuje příklad výpočtu pro střídu 2 a 1009 prefixů. Funkce report_memory vrátí mimo jiné pro tento vstup (různost

podle použitých prefixů) 'prefix_pointers': 1009, 'child_pointers': 6543. Podle vzorce výše zjistíme velikost všech uzlů: ((1009 + 6543) * 4 + 0,5 * 6544) / 1000 = 33,48 kB.

Výpočet velikosti uzlů s nejmenšími ukazateli: ((počet odkazů na prefix * velikost nej. ukazatele na prefixy v bitech + počet odkazů na další uzel * velikost nej. ukazatele na uzly v bitech) / 8 + X B za jednu bitmapu * počet uzlů) / 1000 = nejmenší velikost všech uzlů (celého stromu) [kB]. Následuje příklad výpočtu pro střídu 2 a 1009 prefixů. Funkce report_memory vrátí (mimo jiné) pro tento vstup (různost podle použitých prefixů) 'prefix_pointers': 1009, 'child_pointers': 6543. Podle vzorce na začátku tohoto odstavce můžeme zjistit nejmenší velikost uzlů: ((1009 * 10 + 6543 * 13) / 8 + 0,5 * 6544) / 1000 = 15,17 kB. Z výpočtu můžeme usoudit, že při zvýšení počtu 'prefix_pointers' a 'child_pointers', klesá rozdíl mezi celkovou pamětí (kvůli velikost nej. ukazatele, blížící se 32 bitům) pro ukazatel 32 bitů a nejmenší ukazatel.

Protože se uzly skládají z bitmapy a ukazatelů, můžeme vhodně zvolenou střídou a velikostí ukazatele ušetřit nemalé množství paměti. Zatímco při střídě 4 potřebuje bitmapa 16 bitů (2 B) u střídy 8 je to již 256 bitů (8 B). Tento neadekvátní nárůst bitmapy způsobuje velké plýtvání paměti, jak ukazuje obrázek 5.11.

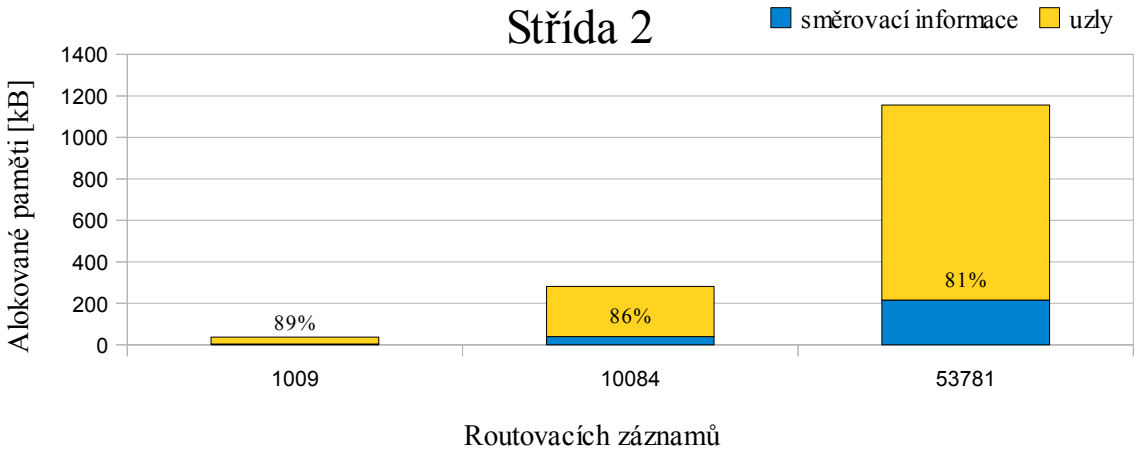
Celková paměť reprezentuje součet velikosti uzlů se směrovacími informacemi. Podle výše uvedených postupů vznikla následující tabulka, která ukazuje paměťové nároky, nad různým množstvím prefixů, pro střídy 2, 4, 8. Střída 16 nemohla být kvůli svým obrovským paměťovým nárokům zařazena do testů.

Lulea		Uzly		Celková paměť	
Směrovací informace	Střída	Počet uzlů	Velikost uzlů [kB]	Ukazatel 32 bitů [kB]	Nejmenší ukazatel [kB]
1009 adres / 4,04 kB	2	6544	33,48	37,52	19,21
	4	3381	24,32	28,36	17,13
	8	1705	65,41	69,46	62,21
10084 adres/ 40,34 kB	2	44358	240,79	281,14	164,11
	4	23203	180,17	220,52	143,06
	8	12666	496,61	536,95	480,52
53781 adres / 215,12 kB	2	155082	940,83	1155,95	717,52
	4	82759	740,12	955,24	632,62
	8	43374	1795,67	2010,79	1763,03

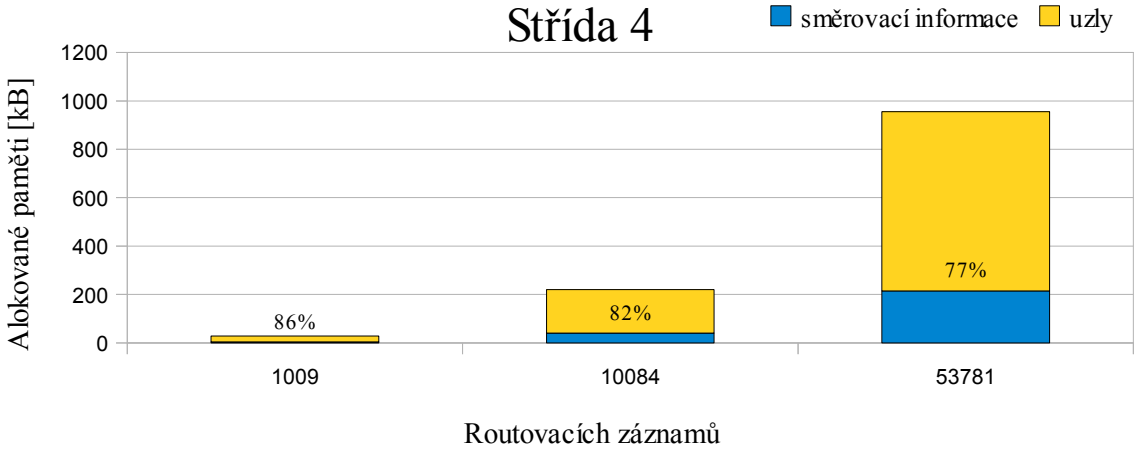
Tabulka 5.6: Paměťové nároky Lulea Compressed Tries.

Z tabulky byly vytvořeny obrázky pro jednotlivé střídy, nad všemi směrovacími informacemi. Obrázky 5.7 – 5.9 zobrazují velikost alokované paměti pro celou strukturu s ukazateli 32 bitů. Tyto obrázky slouží také k demonstraci podílu „užitečných“ (směrovací informace) a „neužitečných“ (uzly) informací. Pro přehlednost jsou v obrázcích uvedena i konkrétní procenta neužitečných informací. U střídy 4 nalezneme nejlepší poměr nutných (směr. info.) / nadbytečných (uzly)

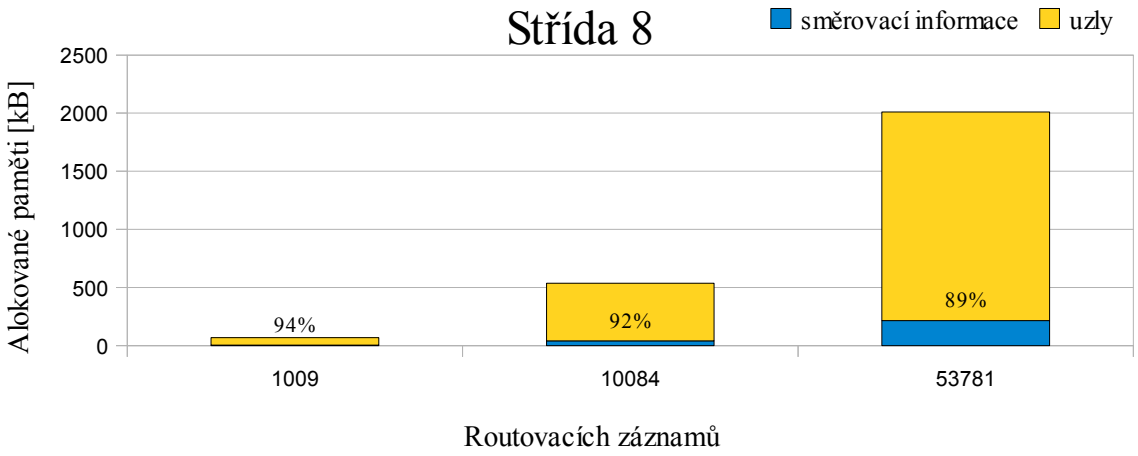
informací, jak ukazuje obrázek 5.8. S větším počtem směrovacích informací je tento poměr více patrný.



Obrázek 5.7: Velikost alokované paměti pro střídu 2 s rozdělením na směr. informace a uzly.



Obrázek 5.8: Velikost alokované paměti pro střídu 4 s rozdělením na směr. informace a uzly.



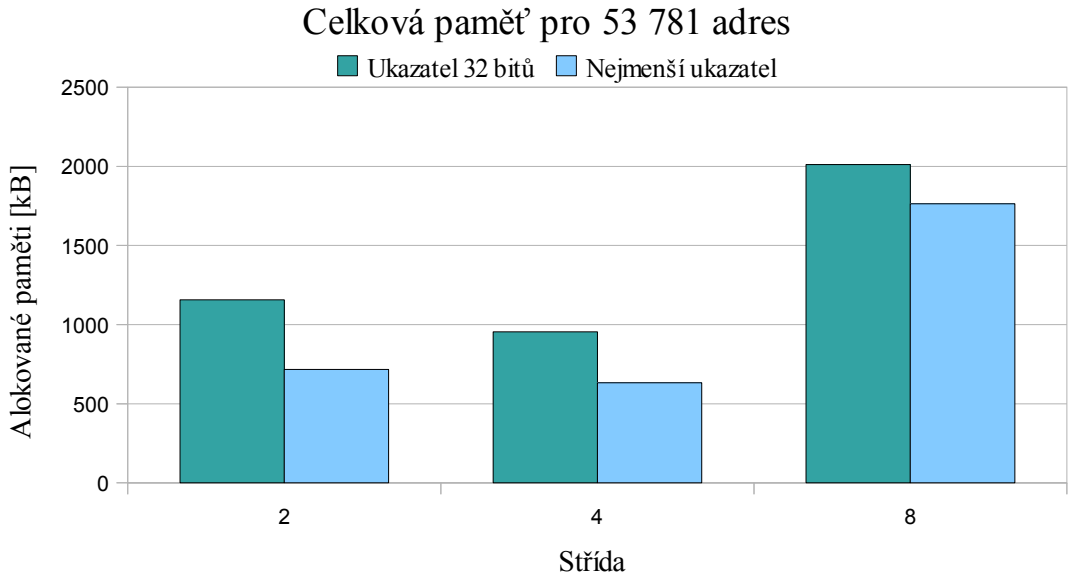
Obrázek 5.9: Velikost alokované paměti pro střídu 8 s rozdělením na směr. informace a uzly.

Následuje průměrný počet bitů 1 v bitmapě na jeden uzel. Počet bitů 1 narůstá se zvětšující se střídou, protože se zvyšuje velikost bitmapy a tím pádem je větší pravděpodobnost uložení bitů 1. Taktéž počet bitů 1 narůstá s přibývajícími prefixy, protože je vice prefixů/odkazů na uložení.

Počet prefixů	1009	10084	53781
Střída 2	1,15	1,23	1,39
Střída 4	1,3	1,44	1,74
Střída 8	1,59	1,8	2,35

Tabulka 5.10: Průměrný počet bitů 1 v bitmapě.

Obrázek 5.11 ukazuje celkovou paměť pro nejmenší ukazatel a pevný 32 bitový ukazatel. Z grafu vyčteme, že největší úspory paměti dosáhneme u střídy 4, která obsahuje nejmenší poměr nadbytečných dat. U střídy 2 nemůžeme moc ušetřit, protože uzel obsahuje příliš malé pole a tím pádem bitmapa redukuje jenom malé množství prefixů. V porovnání dopadla nejhůře střída 8, jelikož neúměrně narostla velikost bitmapy.



Obrázek 5.11: Velikost celkové alok. paměti pro pevné a minimální ukazatele, nad 53 781 prefixy.

5.1.3 Binární vyhledávání na intervalech (BSI)

Paměťové nároky této metody určuje velikost tabulky, kterou vypočteme ze dvou hodnot. Nejprve musíme zjistit velikost jednoho záznamu v tabulce, kterou poté jednoduše vynásobíme počtem řádků tabulky. Počet řádků získáme od funkce report_memory. Podle principu algoritmu zmíněného v 3.4 můžeme usoudit, že je maximální počet záznamů uložených v tabulce dvojnásobkem počtu vstupních prefixů. V reálném provozu lze ale očekávat, že výsledné číslo bude o trochu menší, protože některé záznamy se shodnou hodnotou (indexem) budou reprezentovány jako jeden.

Jeden záznam tabulky obsahuje:

- hodnotu o velikosti 4 B
- odkaz na prefix o velikosti 4 B

Jeden záznam má tedy konkrétní velikost 8 B. Celá tabulka má pak velikost rovnou počtu řádků tabulky * 8 B. Celkovou paměť získáme sečtením velikosti řádků (velikost celé tabulky) a směrovacích informací.

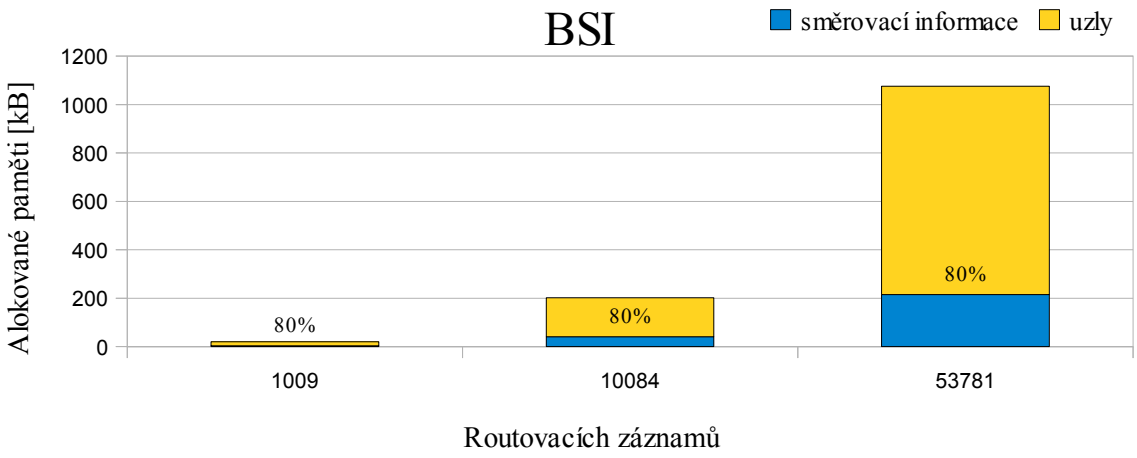
Velikost řádků s nejmenším ukazatelem spočteme jako: (4 B pro hodnotu + počet bitů nej. ukazatele / 8) * počet řádků tabulky / 1000 = nejmenší velikost řádků tabulky [kB]. Ku příkladu pro 1009 prefixů a 2018 řádků tabulky dostaneme: (4 + 10 / 8) * 2018 /1000 = 10,59 kB.

Podle výše uvedených zásad vznikla tabulka 5.12, která ukazuje paměťové nároky, nad různým množstvím prefixů.

BSI	Řádky		Celková paměť	
Směrovací informace	Počet řádků tabulky	Velikost řádků [kB]	Ukazatel 32 bitů [kB]	Nejmenší ukazatel [kB]
1009 adres / 4,04 kB	2018	16,14	20,18	14,63
10084 adres/ 40,34 kB	20168	161,34	201,68	156,31
53781 adres / 215,12 kB	107562	860,5	1075,62	860,49

Obrázek 5.12: Paměťové nároky Binárního vyhledávání na intervalech.

Grafické znázornění celkové paměti s ukazatelem 32 bitů nalezneme na obrázku 5.13. V obrázku jsou pro přehlednost zobrazena procenta nadbytečných informací (uzlů).



Obrázek 5.13: Paměťové nároky BSI s rozdělením na uzly a směrovací informace.

5.1.4 Binární vyhledávání na prefixech (BSP)

Podle sekce 3.5 víme jaké prvky musí obsahovat uzel stromu. Pojďme si nyní ukázat jaké mají tyto prvky teoretickou velikost:

- 4 B pro jeden prefix/odkaz

- 1 bit pro značku delšího prefixu
- 4 B pro odkaz na LPM prefix

Dále zmíním výpočty paměťových nároků, které uvedu dvakrát, v závislosti na velikosti ukazatele (sekce 5.1.1).

Výpočet velikosti uzlů: ((počet vstupních prefixů + počet uzlů - 1) * 4 B) + počet vstupních prefixů / 8 + počet odkazů na LPM prefix * 4 B) / 1000 = velikost všech uzlů (celého stromu) [kB]. Následuje příklad výpočtu pro 1009 prefixů. Funkce report_memory vrátí mimo jiné pro tento vstup (různost podle použitých prefixů) 'nodes': 14, 'count_lpm_mark': 8509. Podle návodu výše zjistíme velikost všech uzlů: ((1009 + 14 - 1) * 4 B) + 1009 / 8 + 8509 * 4 B) / 1000 = 38,25 kB.

Výpočet velikosti uzlů s nejmenšími ukazateli: ((počet vstupních prefixů * velikost nej. ukazatele na prefixy v bitech + (počet uzlů – 1) * velikost nej. ukazatele na uzly v bitech) / 8 + počet vstupních prefixů / 8 + počet odkazů na LPM prefix * velikost nej. ukazatele na prefixy v bitech / 8) / 1000 = nejmenší velikost všech uzlů (celého stromu) [kB]. Následuje příklad výpočtu pro 1009 prefixů. Funkce report_memory vrátí mimo jiné pro tento vstup (různost podle použitých prefixů) 'nodes': 14, 'count_lpm_mark': 8509. Podle návodu ze začátku tohoto odstavce zjistíme nejmenší velikost všech uzlů: ((1009 * 10 + (14 – 1) * 4) / 8 + 1009 / 8 + 8509 * 10 / 8) / 1000 = 12,03 kB. Z výpočtu můžeme usoudit, že nejvíce paměti zabírají odkazy na LPM prefix, poněvadž jich je mnoho.

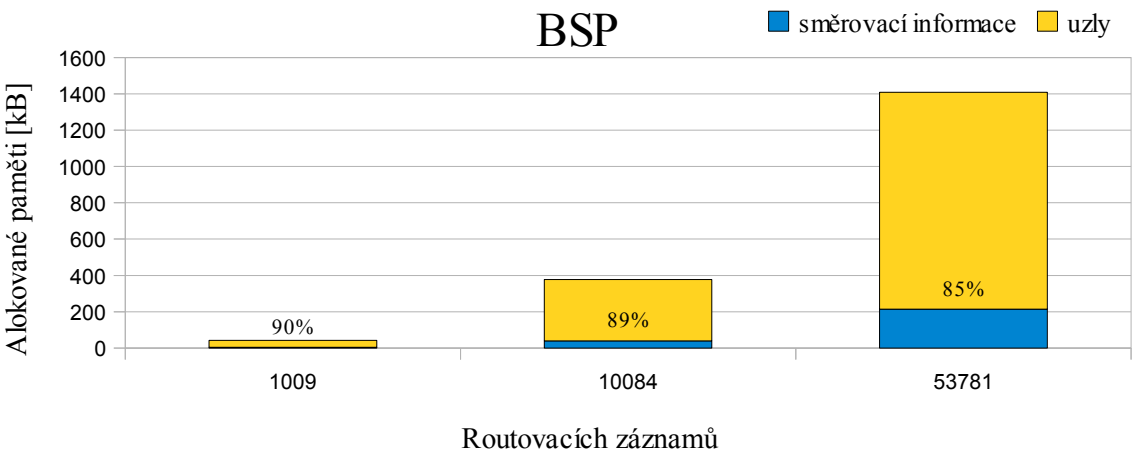
Protože se uzly skládají převážně z ukazatelů, můžeme vhodně zvolenou velikostí ukazatele ušetřit značné množství paměti.

Celková paměť reprezentuje součet velikosti uzlů se směrovacími informacemi. Podle výše uvedených postupů vznikla následující tabulka, která ukazuje paměťové nároky, nad různým množstvím prefixů.

BSP	Uzly		Celková paměť	
Směrovací informace	Počet uzlů	Velikost uzlů [kB]	Ukazatel 32 bitů [kB]	Nejmenší ukazatel [kB]
1009 adres / 4,04 kB	14	38,25	42,29	16,07
10084 adres/ 40,34 kB	19	337,24	377,58	188,57
53781 adres / 215,12 kB	23	1192,99	1408,11	814,94

Tabulka 5.14: Paměťové nároky BSP.

Vypočtená velikost potřebné paměti odpovídá do jisté míry ostatním algoritmům. V tabulce je však vidět, že tato metoda vytváří mizivé množství uzlů oproti ostatním algoritmům. Díky tomu přistupuje do paměti méně častěji, což potvrzuje obrázek 5.17. Grafické znázornění potřebné paměti nalezneme v obrázku 5.15 (pro přehlednost jsou zobrazena procenta nadbytečných informací (uzlů)).



Obrázek 5.15: Paměťové nároky BSP s rozdělením na uzly a směrovací informace.

5.2 Počet přístupů do paměti v nejhorším případě

5.2.1 Teoretický závěr

Praktické vyhodnocení maximálního počtu přístupů do paměti je silně závislé na vstupní množině prefixů, a tak neposkytuje při určitých vstupních množinách věrohodné výsledky. Proto v této podkapitole zmíním teoretické závěry nad maximálním počtem přístupů do paměti. Dále budu vycházet z poznatku, že prefix pro IPv4 může nabývat délky 1 až 32, tedy 32 možných délek. Toto je důvod, proč je praktické vyhodnocení neúčinné. Respektive by vstupní množina prefixů musela obsahovat všech 32 možných délek.

Většina algoritmů, založená na principu vyhledávání nad binárním stromem, potřebuje znát k určení maximálního počtu přístupů do paměti, délku nejdelšího vstupního prefixu. Který je v našem případě 32. Následují teoretické domněnky o počtech přístupů do paměti v nejhorším případě pro jednotlivé algoritmy:

- Trie – záleží na hloubce stromu, tedy 32 přístupů.
- CPE a Lulea – záleží na použité střídě. Při použití střídy 4 dostaneme 8 přístupů (32/4).
- Binární vyhledávání na intervalech – záleží na množství kroků, které musíme v tabulce vykonat, než nalezneme výsledek. Záleží tedy jakým způsobem procházíme tabulku. Již podle názvu můžeme usoudit, že tabulku budeme procházet za pomoci binárního vyhledávání (tzv. metoda půlení intervalu). Tento způsob vyhledávání potřebujeme maximálně $\log_2 N$ přístupů. Přičemž N reprezentuje počet řádků tabulky.

- Binární vyhledávání na prefixech – opět vyhledáváme za pomoci binárního vyhledávání, tudíž je maximální počet přístupů roven vzorci $\log_2 N$. Kde N představuje počet uzlů stromu. Pro 32 uzlů dostaneme 5 přístupů.

5.2.2 Praktické testy

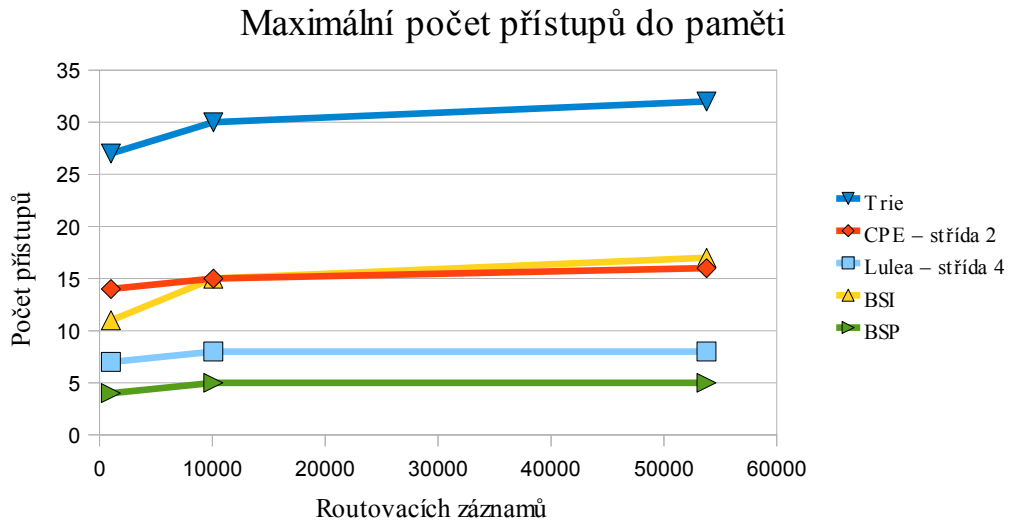
Nyní si ukážeme reálné výsledky, nad stejnou množinou routovacích záznamů jako u sekce 5.1. K reálným výsledkům dojdeme na základě teoretických závěrů a hodnot z testů. Testy zjistí hloubku stromu či velikost tabulky, dále podle teoretických závěrů vypočteme konečné hodnoty, které ukazuje tabulka 5.16. Z tabulky je patrné, že se čísla mírně liší od teoretických závěrů, především pak u malé vstupní množiny. Tento rozdíl byl již vysvětlen v předchozí části (5.2.1).

U metod CPE a Lulea byla vybrána hodnota střídy podle nejmenších paměťových nároků. Do tabulky je pro účely porovnání zahrnuta i metoda Trie.

Počet prefixů	1009	10084	53781
Trie	27	30	32
CPE – střída 2	14	15	16
Lulea – střída 4	7	8	8
BSI	11	15	17
BSP	4	5	5

Tabulka 5.16: Maximální počet přístupů do paměti v závislosti na množině prefixů.

Na základě tabulky 5.16 vznikl následující obrázek, který porovnává jednotlivé algoritmy mezi sebou. Do grafu je pro názornost vložena i výchozí metoda Trie.



Obrázek 5.17: Porovnání algoritmů z pohledu počtu přístupů do paměti v nejhorším případě.

6 Porovnání algoritmů

Kapitola představuje vyhodnocení jednotlivých algoritmů, jejichž paměťové nároky byli uvedeny v páté kapitole. Každé porovnání je uvedeno pro pevný a minimální ukazatel zvlášť. U algoritmů CPE a Lulea byly vybrány střídy, které zabírají nejméně paměti.

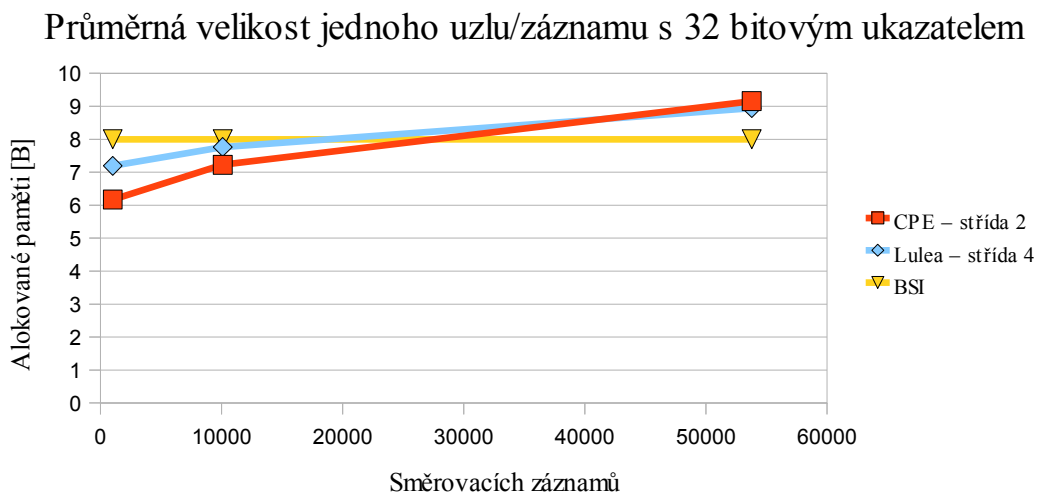
6.1 Velikost jednoho uzlu/záznamu

Na základě dat z tabulek uveřejněných v 5 kapitole vznikly tabulky 6.1 a 6.3, ukazující paměťové nároky jednoho uzlu/záznamu, pro pevný a minimální ukazatel. Tabulky byly pro lepší porozumění vypracovány i do grafické podoby, kterou představují obrázky 6.2 a 6.4, ovšem již bez BSP, protože obsahuje příliš veliké hodnoty do grafu.

Na obrázcích je vidět narůstající velikost uzlu s přibývajícím počtem záznamů, to je logické, poněvadž se musí do uzlů ukládat čím dál více dat. U metody BSI můžeme pozorovat pro pevný ukazatel velice dobrý poměr potřebné paměti na jeden záznam tabulky.

Prefixů	1009	10084	53781
CPE – střída 2 [B]	6,17	7,22	9,16
Lulea – střída 4 [B]	7,19	7,77	8,94
BSI [B]	8	8	8
BSP [B]	2732,15	17749,29	51868,98

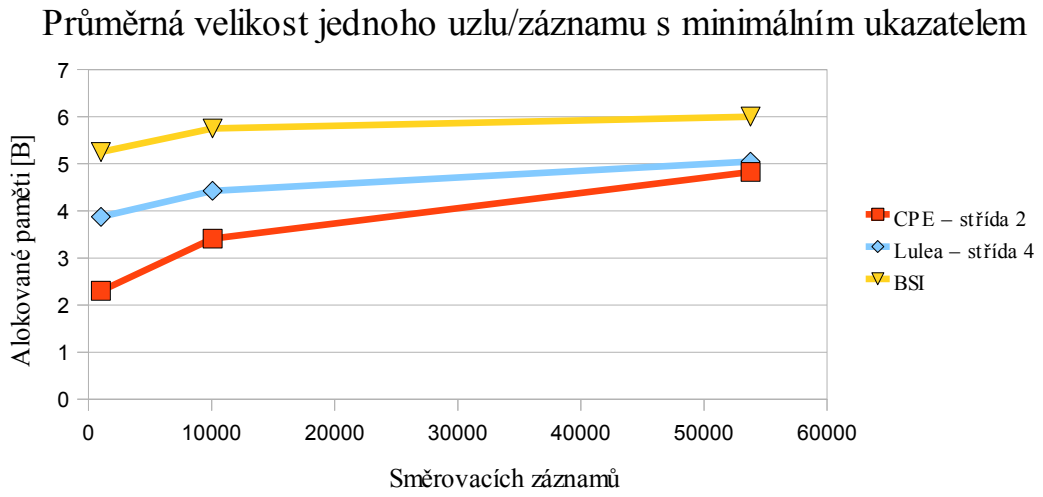
Tabulka 6.1: Průměrná velikost jednoho uzlu/záznamů v závislosti na počtu prefixů pro pevný 32 bitový ukazatel.



Obrázek 6.2: Grafická podoba tabulky 6.1 bez BSP.

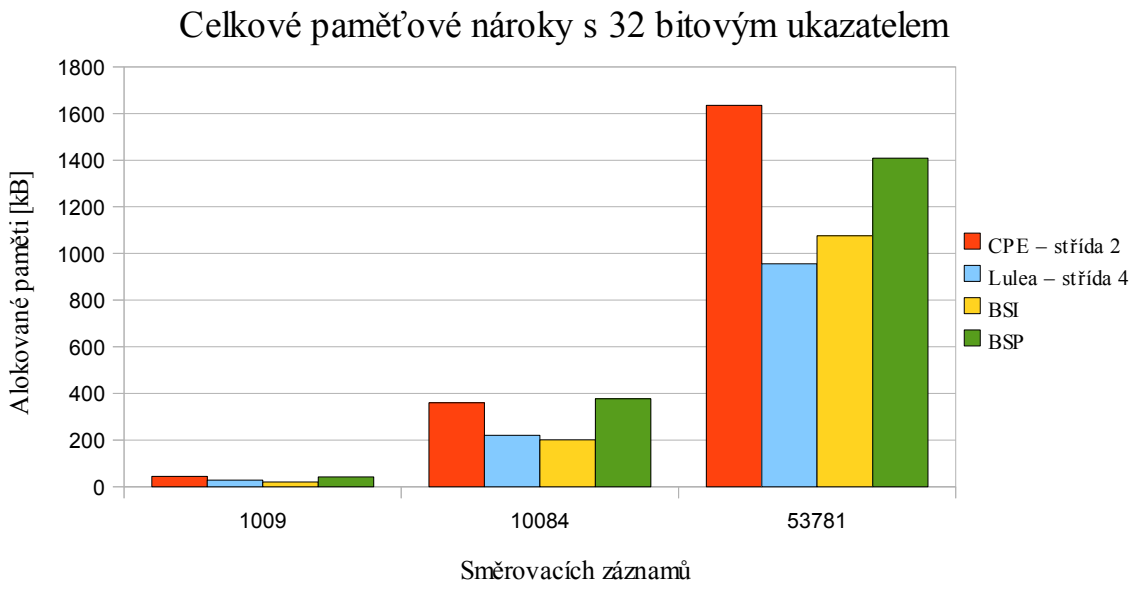
Prefixů	1009	10084	53781
CPE – střída 2 [B]	2,3	3,41	4,83
Lulea – střída 4 [B]	3,87	4,43	5,04
BSI [B]	5,25	5,75	6
BSP [B]	859,29	7801,57	26079,32

Tabulka 6.3: Průměrná velikost jednoho uzlu/záznamů v závislosti na počtu prefixů pro minimální ukazatel.



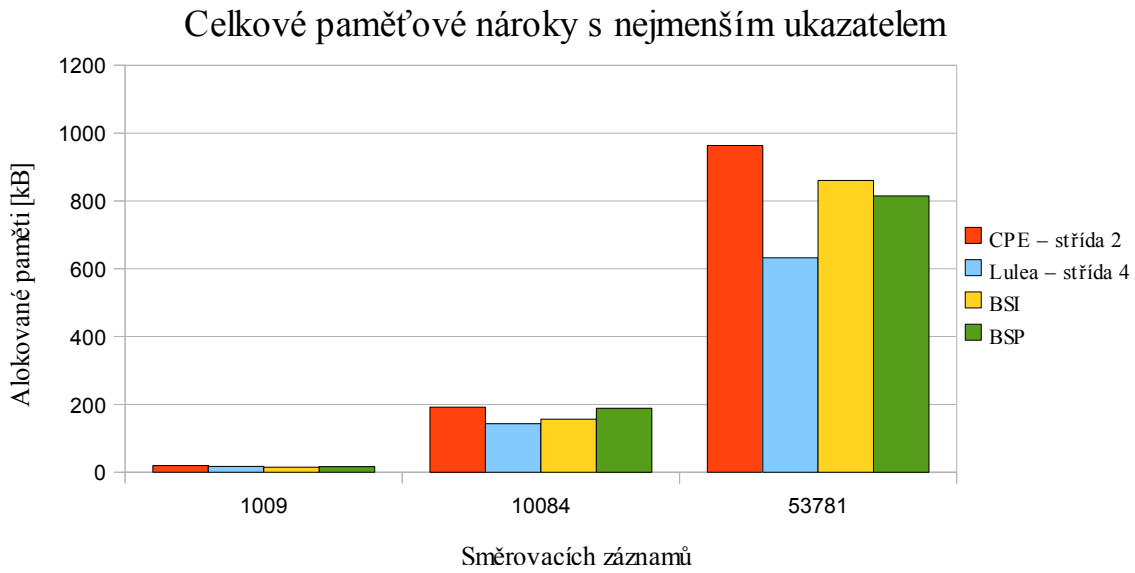
Obrázek 6.4: Grafická podoba tabulky 6.3 bez BSP.

6.2 Celková velikost



Obrázek 6.5: Porovnání celkové paměti pro pevný 32 bitový ukazatel.

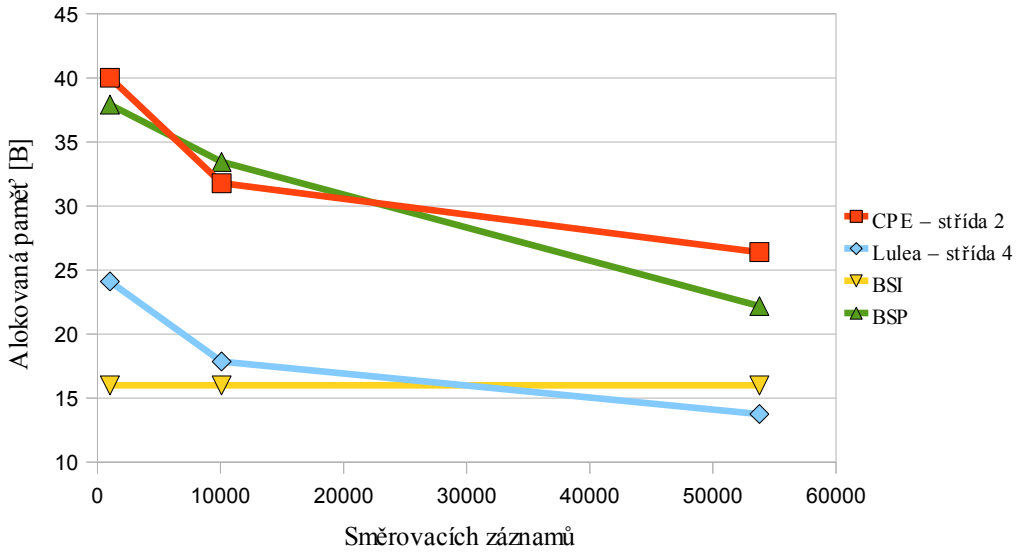
S minimálním ukazatelem jsou výsledky porovnání poněkud odlišné. Z obrázku 6.5 můžeme vycítit malé paměťové nároky BSI, ovšem při použití minimálního ukazatele máme u BSI jenom jedno místo, kde můžeme ušetřit. Naproti tomu u ostatních algoritmů používáme ukazatele častěji, a tím docílíme větší úspory paměti, jak je patrné u BSP, kde se nachází veliké množství ukazatelů.



Obrázek 6.6: Porovnání celkové paměti pro minimální ukazatel.

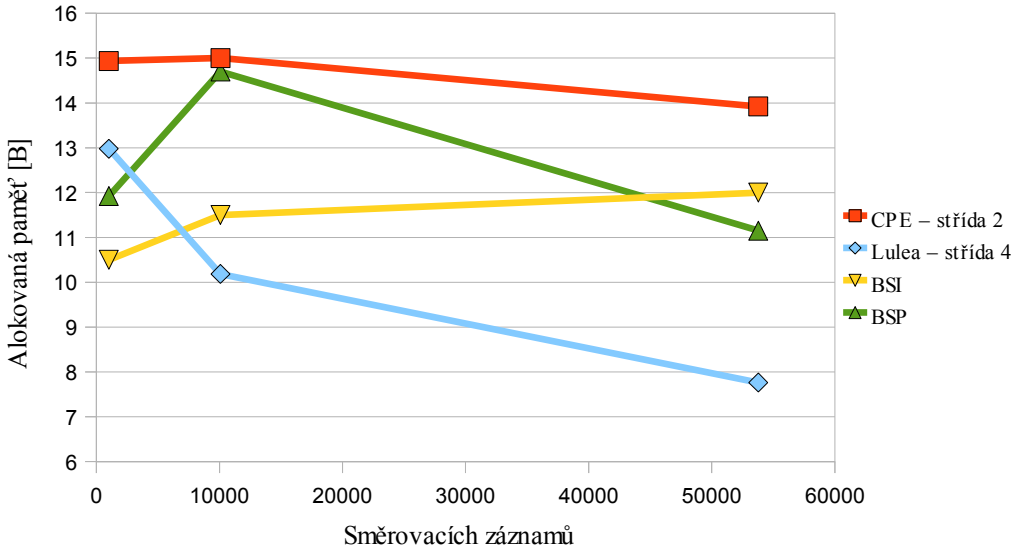
6.3 Potřebná paměť na jeden směrovací záznam

Zajímavým ukazatelem efektivity algoritmu je mimo jiné i průměrná velikost paměti na jeden směrovací záznam. Hodnoty alokované paměti pozvolna klesají v důsledku umění algoritmů lépe redukovat potřebnou paměť pro větší množství směrovacích informací.



Obrázek 6.7: Potřebná paměť na jeden směrovací záznam s pevným ukazatelem.

S použitím minimálního ukazatele vzniká zajímavý stav, kdy hodnoty alokované paměti v určitých místech grafu stoupají. Tento jev se vynořuje při rychlejším nárůstu potřebné paměti než počtu směrovacích záznamů.



Obrázek 6.8: Potřebná paměť na jeden směrovací záznam s minimálním ukazatelem.

7 Závěr

Práce shrnuje paměťové nároky a maximální počet přístupů do paměti vybraných LPM algoritmů. Aby mohlo vzniknout toto shrnutí, musela být nastudována problematika vyhledávání nejdelšího shodného prefixu v IP sítích. Dále pak byly nastudovány, rozebrány a naimplementovány vybrané LPM algoritmy. Nakonec byla, podle výpočtů nad daty z testů, zjištěna paměťová náročnost jednotlivých algoritmů.

Již z kapitoly 3, ve které byly popsány principy jednotlivých algoritmů, můžeme usoudit jak budou algoritmy paměťově náročné. Jeden ze závěrů, které můžeme učinit pro algoritmy CPE a Lulea je, že díky používání bitmapy u Luley ušetříme velké množství paměti oproti CPE. U algoritmu BSI můžeme dojít k závěru, že má každý záznam tabulky konstantní velikost. Díky tomu disponuje BSI velice dobrým poměrem potřebné paměti k použitým prefixům. Za zmínku stojí i algoritmus BSP, který může nalézt uplatnění všude tam, kde je zapotřebí minimálního počtu přístupů do paměti. Musíme však počítat s vyššími paměťovými nároky.

Tyto závěry potvrdila kapitola 5, která ukázala reálné paměťové nároky jednotlivých algoritmů. Můžeme zde pozorovat plýtvání paměti u každého algoritmu, jelikož tyto algoritmy potřebují ke své práci velké množství podpůrné paměti. U algoritmů CPE a Lulea byly zjištěny nejúspornější střídy, konkrétně 2 pro CPE a 4 pro Lulea. U paměťových nároků se dále ukázalo, že použití pevného či minimálního ukazatele vede k odlišným paměťovým nárokům. Kapitola 5 se kromě paměťových nároků zabývala také nutností přistupovat do paměti. Zde dopadly velice dobře všechny algoritmy, oproti základní metodě Trie. Avšak úplně nejlépe dopadla metoda BSP, která díky minimálnímu počtu uzlů přistupuje do paměti bezkonkurenčně nejméně.

Následné porovnání jednotlivých algoritmů proběhlo v kapitole 6. Nejmenší paměťové nároky byli zjištěny u Luley se střídou 4, jenž má nejmenší poměr nadbytečných informací, díky tomu alokuje nejméně paměti.

Hlavní přínos práce je ve vytvoření unifikovaného prostředí pro porovnání daných LPM algoritmů. Tyto poznatky budou použity při výběru, návrhu a realizaci algoritmů do hardwaru v rámci projektu ANT, jehož je práce součástí.

Literatura

[1] Internet Protocol In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation,[cit. 2010-04-24]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Internet_Protocol>.

[2] PERLMAN, Radia. *Interconnections: bridges, routers, switches, and internetworking protocols*. Second Edition. [s.l.] : Addison-Wesley, 2000. Chapter 9. Network Layer Addresses, s. 189-220. ISBN 0201634481.

[3] Routování In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, ,[cit. 2010-04-25]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Routování>.

[4] PERLMAN, Radia. *Interconnections: bridges, routers, switches, and internetworking protocols*. Second Edition. [s.l.] : [s.n.], 2000. Chapter 13. Fast Packet Forwarding, s. 347-366. ISBN 0201634481.

[5] TOBOLA, Jiří. *Pojednání k tématu Dizertační práce*. [s.l.], 2009. 26 s. Odborná práce. Vysoké Učení Technické v Brně, Fakulta Informačních Technologií.

[6] V. Srinivasan, G. Varghese, "Faster IP Lookups using Controlled Prefix Expansion," Proc. SIGMETRICS '98.

[7] LAMPSON, Butler; SRINIVASAN, Venkatachary; VARGHESE, George : IP lookups using multiway and multicolumn search. In *IEEE/ACM Transactions on Networking (TON)*. [s.l.] : [s.n.], 1999. s. 324-334. ISSN 1063-6692.

[8] KIM, Kun Suk; SAHNI, Sartaj : IP Lookup By Binary Search On Prefix Length. In *Proceedings of the Eighth IEEE International Symposium on Computers and Communications*. [s.l.] : [s.n.], 2003. s. . ISBN 0-7695-1961-X.

[9] WALDVOGEL, Marcel; VARGHESE, George; TURNER, Jon; PLATTNER, Bernhard : Scalable high speed IP routing lookups. In *ACM SIGCOMM Computer Communication Review*. [s.l.] : [s.n.], 1997. s. 25 - 36. ISSN 0146-4833.

[10] *Python Programming Language -- Official Website* [online]. Copyright © 1990-2010 [cit. 2010-04-27]. About Python. Dostupné z WWW: <http://www.python.org/about/>.

[11] *Accelerated Network Technologies* [online].[cit. 2010-04-30]. Home. Dostupné z WWW: <http://merlin.fit.vutbr.cz/ant/home/index.html>.

[12] *BGP Reports* [online].[cit. 2010-05-07]. AS65000 - BGP Table Statistics. Dostupné z WWW: <http://bgp.potaroo.net/as2.0/bgp-active.html>.

[13] *Wikipedie, otevřená encyklopedie* [online].[cit. 2010-05-07]. Border Gateway Protocol. Dostupné z WWW: <http://cs.wikipedia.org/wiki/BGP>.

Seznam příloh

Příloha 1. DVD se zdrojovými kódy.